DEVELOPMENT OF A SEED COTTON FIBER QUALITY SENSING SYSTEM FOR COTTON FIBER QUALITY MAPPING

A Thesis

by

VINCENT PAUL SCHIELACK III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2011

Major Subject: Biological and Agricultural Engineering

Development of a Seed Cotton Fiber Quality Sensing System

for Cotton Fiber Quality Mapping

DEVELOPMENT OF A SEED COTTON FIBER QUALITY SENSING SYSTEM FOR COTTON FIBER QUALITY MAPPING

A Thesis

by

VINCENT PAUL SCHIELACK III

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	J. Alex Thomasson
	Ruixiu Sui
Committee Member,	Cristine Morgan
Head of Department,	Steve Searcy

December 2011

Major Subject: Biological and Agricultural Engineering

ABSTRACT

Development of a Seed Cotton Fiber Quality Sensing System
for Cotton Fiber Quality Mapping. (December 2011)
Vincent Paul Schielack III, B.S., Texas A&M University
Co-Chairs of Advisory Committee: Dr. J. Alex Thomasson
Dr. Ruixiu Sui

For precision agriculture to work, an automated process to collect spatial-variability data within a field is necessary. Otherwise, data collection is prohibitively expensive and time consuming. Furthermore, to minimize measurement error due to harvesting method, data-collection processes involving normal cotton harvesting and ginning operations must be used.

For the case of cotton, an automated prototype system using image processing to measure the micronaire value of cotton fiber during harvest was designed and built in the laboratory. This system was tested with two image-processing algorithms to identify and remove the effects of objects present in the images that were not cotton fiber, and then measure the reflectivity in three Near-Infrared (NIR) wavebands. Both algorithms yielded similar results when used on seed cotton samples. The reflectivity measurement after removing the effects of foreign matter had a strong relationship to standard micronaire measurements (R^2= 0.73 and 0.74 for the ratio-image and single-image algorithms, respectively) with a root mean squared error (RMSE) of 0.28 and 0.27, respectively. The ratio-image pixel classification method classified an average of 58% of the pixels in an image as "cotton", while the single-image method classified an average of 81% of the pixels in each image as cotton. These results do not show as strong a relationship between micronaire and NIR reflectivity of cotton samples as previous research done with very uniform lint cotton calibration samples. This is attributed to the higher content of foreign matter in seed cotton samples. With higher

trash cotton and fiber that has not yet been cleaned, results obviously are not as good as when using calibration cotton samples. These results indicate the system can be adapted to perform in-situ measurement of cotton fiber quality, specifically micronaire, and enable harvesters to create quality maps of a field automatically to allow better crop management.

ACKNOWLEDGEMENTS

I would like to thank my committee co-chairs, Dr. Thomasson and Dr. Sui, and my committee member, Dr. Morgan, for their guidance and support throughout the course of this research, and their patience during the writing of this manuscript.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. I also want to extend my gratitude to the Fiber and Biopolymer Research Institute in Lubbock, Texas for providing time and equipment for quality measurement, as well as the IMPACT center for allowing this research to disrupt their harvest even more than the rain. Special thanks go to Dr. Ge, for answering the countless questions I asked of him during all hours of the day, A.J. Sjolander, for being brave enough to help hand sample cotton from a moving harvester, to Amanda Boswell, and to Jessica Hammons, for motivation to finish this project as well as providing much needed distractions so I could stay focused.

Finally, thanks to my mother and father for their encouragement and support throughout not only this research, but also my prior and continued education.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

INTRODUCTION

The long term sustainability of any farming operation is dependent on the profit that the crop can bring. The profitability of a given cotton crop is determined by two major factors, the total yield and the quality of the fiber. While higher yield will bring in a higher profit, the quality of the cotton, including properties like the micronaire value, determines the price at which it will sell. By increasing quality, a harvested crop can sell at a higher price. Crop quality is determined by the measureable properties of the fiber, grain, or fruit that is harvested. As the properties of the crop become more desirable, further processing (such as more trash removal) can be done more cost-effectively. Cotton, for example, must be ginned after it is harvested, and is then typically spun and dyed. Fiber properties, such as length, strength, and fineness determine how efficiently the fiber can be spun, while maturity will affect dye uptake and fabric appearance.

Cotton fiber quality varies across farm fields (Johnson et al. 2002, Sassenrath et al. 2005, Ge 2007, Sjolander 2009, Ge et al. 2009). Currently, the only practical way to track the variability of cotton quality within a field is by tracking the harvest locations of each module, and then assigning an average quality based on the measured properties of each bale from that module to the locations in the field from which it came. Research has shown that this is a feasible approach (Ge 2007, Sjolander 2009), albeit not yet commercialized. However, a quality variability map generated in this manner does not match the resolution of other variability maps, such as maps of yield, elevation and soil electrical conductivity. To generate a higher resolution fiber quality map, the only current option is to take a great number of samples manually, which would be prohibitively time and labor intensive. If an automated fiber quality sensor were available for use on-board a harvester, fiber quality measurement could be integrated into the harvesting process, and high-resolution fiber quality variability maps could be readily generated.

This thesis follows the style of *Transactions of the American Society of Agricultural and Biological Engineers.*

Cotton Quality

Cotton quality is measured in terms of a number of different fiber properties, each of which will have an effect on textile processing. These properties are measured with 230-g (8-oz.) samples from each ginned cotton bale (one 115-g sample from each side of the bale) to classify the cotton based on HVI (High Volume Instrument) classification. The following properties fall within the scope of HVI classification (Cotton Inc., 2010):

- Fiber Length
- Length Uniformity
- Fiber Strength
- Micronaire
- Color Grade
- Trash Content

Fiber Length

The length of cotton fiber is mainly determined by the variety, but harsh conditions during growth and rigorous ginning and cleaning can cause the fibers to shorten. Fiber length has an influence on yarn fineness, strength, and uniformity, and longer fibers will make the spinning process more efficient. Fiber length is commonly measured as the average fiber length of the longer half of a sample, or the upper half mean length (UHML) (Cotton Inc., 2010). Generally, longer fibers are more desirable.

Length Uniformity

Length uniformity is a ratio of the mean length of a sample of cotton fiber to the UHML, expressed as a percentage. A length uniformity of 100% would mean that all of the fibers in the sample have the same length (a practical impossibility). High length uniformity allows yarn to be spun more evenly and increases the strength of the yarn, as well as making the spinning process more efficient. Low length uniformity is normally

associated with high short fiber content (SFC), and will be reflected in the lower-quality yarn produced (Cotton Inc., 2010).

Fiber Strength

Yarn strength has a direct relationship with fiber strength. Stronger fibers produce stronger yarn. In HVI classification, fiber strength is measured on the same samples that are used to measure fiber length. Testing strength is destructive, as it is a measurement of how much tensile force (g) is required to break a bundle of fibers one tex unit in size. One tex unit is equal to the weight (g) of 1,000 m of fiber from the given sample. Higher-strength fibers tend to be more durable and less likely to break in the ginning and manufacturing processes (Cotton Inc., 2010).

Color Grade

Color grade is a measurement of the reflectance (Rd) and yellowness (+b) of cotton fiber. Variations in color grade mean variation in the fiber's color after dyeing. A low color grade is also related to an increased probability of lower efficiency during processing (Cotton Inc., 2010).

Trash Content

A trash content measurement is the measurement of any matter within the fiber sample that is not cotton fiber. Trash includes material like seed coat fragments, leaf particles, and bark. Trash content is dependent largely on the variety of cotton and the harvesting method used. Even after careful harvesting and ginning with the most rigorous cleaning methods available, there will be residual trash content in the cotton lint. This so-called "leaf" content is waste and has an associated cost for removal before spinning (Cotton Inc., 2010).

Micronaire

The maturity and fineness of cotton fiber are expressed with a unit called micronaire. Fiber maturity is a measure of the fiber wall thickness and can be described as a ratio of this cell wall thickness (cotton fibers are individual cells) to the diameter of the fiber. Immature fibers have thinner cell walls compared to mature fibers of the same diameter. Fiber maturity affects the appearance of fabrics, partly because immature fibers absorb less dye than mature fibers. Maturity also has an impact on nep (entanglement) formation of cotton fibers.

Fineness is a measure of the effective outer diameter of cotton fibers. Fineness has an effect on the strength of individual cotton fibers, as fibers with smaller diameters generally require less force to break. However, finer cotton means improved spinning efficiency and yarn strength, because it means more cotton fibers per cross section of yarn. High-micronaire cotton fiber sells for a premium up to a point, but any further increase in the micronaire value reduces the price (Chakraborty and Ethridge, 1999).

Measurement Type

To measure micronaire with an HVI system, a specified mass of cotton fiber is compressed, and, as air is passed through the fiber, the resistance to airflow is measured. This value is converted to units of airflow and used to calculate the micronaire value for the sample. HVI micronaire measurement is an established and accepted method of classifying the fineness and maturity of cotton fiber. However, this method is only applicable in a laboratory setting, as the instrumentation is large, expensive, and requires strictly controlled ambient conditions (ASTM 2011).

By using optical methods to measure the properties of cotton fiber, the size of the instrumentation can be reduced, allowing the system to be portable and potentially mounted on a harvester. Optical methods of fiber quality measurement relate the amount of light reflected by the cotton at specified wavelengths to a desired property based on an established model. These models must first be found by relating cotton fiber reflectance back to measured fiber properties and determining a relationship.

Site Specific Crop Management

SSCM (Site Specific Crop Management) is a field management practice used to optimize production of agricultural crops. The basic principle is that a site within a field is managed based on its specific needs to facilitate improved harvests and reduce environmental impact (Ge, 2007). SSCM involves measuring the field and crop conditions, yield, and quality in order to vary management decisions regarding inputs such as chemical application and irrigation. SSCM data can be used for decision making regarding subsequent cropping seasons or even during the current cropping season with VRT (variable-rate technology) to reduce over- or under-application of (e.g.) fertilizers and pesticides. Using VRT can not only reduce costs by reducing unnecessary chemical use, but it also reduces environmental impacts; as chemicals are applied on an "as needed" basis, the amount of residual chemical left unused by the plant is reduced. For SSCM and VRT to be implemented to improve the overall quality of the fiber in a cotton field, quality-variation maps with a higher resolution than is currently available are necessary. A detailed quality-variation field-map generated during harvest would allow these practices to be used with respect to the historical quality variation within a field. Detailed fiber-quality spatial variability could be used to adjust management practices such as seeding and irrigation rates, as well as aid in showing profit variability when used with yield and cost maps.

Literature Review

Cotton Quality Variation

The quality of cotton has been shown to vary significantly within a field (Johnson et al., 2002, Sassenrath et al., 2005). Among the properties typically measured, micronaire is the most spatially variable (Ge et al., 2008). This variation can be caused by soil factors like EC_a (apparent electrical conductivity) (Ge et al., 2009) and soil moisture (Ge, 2007). VRT can potentially be applied based on EC_a data to change the

seeding rate during planting to optimize revenue from cotton yield and quality with respect to seed input costs (Stanislav, 2010). To monitor the effects of SSCM on the quality of a cotton field over several planting seasons, the quality of the harvested cotton must be mapped back to the location from where it came in the field. Current methods of mapping quality variation are labor intensive and time prohibitive, requiring numerous samples to be collected by hand, and leaving an automated method of sensing and recording fiber quality to be desired.

Previous research on mapping cotton quality has largely required cotton samples to be hand harvested (Johnson et al., 2002, Ge, 2007, Ge et al., 2009) and processed separately from the main mechanical harvest. Manual harvesting of samples not only takes a great deal of time, but it also results in quality measurements that differ from those of the rest of the field due to differing harvest methods (Calhoun et al., 1996, Faulkner et al., 2008).

Fiber Quality Mapping

The values measured in each sample are averaged and assigned to the bale from which they came, but it is not currently possible to track where each bale came from in a cotton field. At a reasonable harvest rate of 1,400 kg/ha (2.5 bales per acre), it takes approximately 2.4 ha. (6 ac.) of cotton to build a module (15 bales). As the module is built, the cotton harvested from this area is mixed within the module builder, and then possibly mixed even more during ginning. An automated wireless module-tracking system was recently developed and tested, and it provides data for mapping module-harvest boundaries (Ge, 2007; Sjolander, 2009). By averaging the quality of the bales produced from a module, the quality data can be mapped back to the field to create a fiber quality map with a "per module" resolution that can be used to improve field management. The automated system included wireless transceivers for communication among cotton-harvest field machines (harvesters, boll-buggies, and module builders), a basket-tilt sensor and load cells to enable automation of signal transmission when cotton is transferred from machine to machine, and RFID for machine-to-machine identification. This system was tested in conjunction with a cotton yield monitor that

included a mass-flow sensor (Thomasson and Sui, 2004) and GPS, so yield variation within the field could also be mapped at a high resolution. This module-tracking system allows HVI measured quality data for each cotton module to be applied back to the location in the field from which it came. While a useful innovation, this method of quality mapping only produces data on a module-by-module basis. Since each module can be built by cotton from very different locations in a field, especially when multiple harvesters and multiple module builders are used, individual cells within the resulting quality maps can be not only of unusual shapes and sizes but also noncontiguous. Higher-resolution quality maps are needed to more efficiently use quality data in SSCM. The most obvious way to achieve higher-resolution quality maps is to measure quality parameters such as micronaire, which exhibits the most spatial variability, on-board the harvester. This system ultimately allowed (potentially multiple) harvesters and boll buggies to communicate with (potentially multiple) module builders while minimizing interaction from the operator, and opened the door to automated quality mapping, but the average quality of the module had to be assigned to every portion of each cotton row that went into the module. This process results in a low-resolution quality map – including (potentially noncontiguous) cells of mixed shapes and sizes – that can be significantly erroneous when comparing cotton quality at any point in the field to the value of the module into which that cotton went. Since a quality map of finer resolution and uniform, contiguous cells is desired, quality must be determined with samples much smaller than an entire module.

Faulkner et al. (2008) reported on cotton fiber-quality variation between harvesting methods. Their research showed that cotton harvested with a picker tended to have higher micronaire values than cotton harvested with a stripper. It is also known that hand-harvesting produces fiber with quality values that differ from machine-harvested cotton. Fiber-quality maps developed in research projects have typically involved hand harvesting cotton at specific points in the field and recording the quality. Since harvesting method has an effect on the final measured quality of cotton lint, it is important to be able to measure fiber quality on samples harvested with the type of

machine being used in the field of interest. Otherwise there would be an additional, harvest-method based, source of error between the map and the measured quality of the module. To reduce this error, a sampling method that operates onboard the harvester is needed.

Thomasson and Shearer (1995) found a relationship between near-infrared (NIR) reflectance and certain cotton quality characteristics in lint cotton. Their models resulted in the following R^2 values: 0.88 for reflectance, 0.85 for yellowness, 0.60 for trash content, 0.96 for micronaire, 0.73 for strength, 0.79 for length, and 0.67 for length uniformity. Sui et al. (2008) showed that NIR reflectance from cotton micronaire calibration standards correlated very well (R^2=0.99) with their micronaire values. This method, which included the use of an NIR camera, was used strictly on lint cotton and requires modification to work with seed cotton in order for fiber quality to be determined during harvest. Unlike the lint cotton standards used, machine harvested seed cotton contains large amounts of visible foreign matter as well as cotton seeds.

In order to measure cotton quality on the harvester and ensure that harvested seed cotton is being sampled and measured in near real time, an automated sampling system that captures cotton in the duct between the harvester head and the basket could be used. Sassenrath et al. (2005) developed a hand-operated valve-type device to remove samples from a spindle-type harvester duct so they could be hand labeled according to field position and later measured for quality. They subsequently automated this process (Sassenrath et al. 2006), thereby increasing the sampling rate. It is desirable to find a way to collect such a sample, determine the GPS coordinates and quality immediately, and then release the sample back into the flow of cotton.

A system capable of measuring cotton fiber quality during harvest would be very useful to agricultural operations that are making use of emerging VRT and SSCM technologies. Currently, field maps can be generated that show yield variability and cost variability within the field as it is grown and harvested. A quality map, when used along with a yield map, price schedule, and cost map can ultimately provide the farmer with a profit map, an important tool in optimizing profit on a site-specific basis. Furthermore,

determining the quality of seed cotton during the harvest provides for high-resolution maps, meaning that site-specific optimization can potentially be done on a small scale. Fiber-quality maps could also provide farmers, ginners, and the market with more information about each cotton harvest before the modules leave the farm, potentially enabling an increase in marketing efficiency.

Objectives

The long-term goal of this research is to develop a system that does the following:

1. samples seed cotton from continuous airflow in a harvester duct
2. presses the sample against a window with a known force
3. automatically collects and stores images of the sample in appropriate spectral wavebands
4. records GPS coordinates of each sample
5. releases the sample back into the airflow
6. uses image analysis on the stored seed cotton sample images to determine fiber quality by measuring reflected energy on the cotton fiber portion only, removing the effects of foreign matter in the sample image
7. combines quality measurements and GPS coordinates to create a quality map of the field harvested.

The specific objectives of the current project relate to functions 3 and 6 above. They are laid out in more detail as follows:

1. Design a system that uses image processing of NIR images to estimate the HVI micronaire value of machine-harvested seed cotton.
2. Build a prototype system that estimates HVI micronaire of machine-harvested seed cotton in a laboratory setting. This prototype has two basic requirements:
    a. A hardware apparatus that can accept a seed cotton sample and acquire images with minimal user assistance.
    b. Software that can control the hardware to collect image data from the cotton samples and process the images to calculate a micronaire value.

3. Conduct a test in which
    a. The relationship between HVI micronaire and reflectivity of fiber in machine-harvested seed cotton is found.
    b. The relationship between HVI micronaire and reflectivity of machine-harvested lint cotton is found.
    c. The function and accuracy of the sensing system are evaluated.

SOLUTION PROCEDURE

Development of a seed-cotton fiber-quality sensor for the purpose of measuring fiber quality during cotton harvest requires the design of a system that can measure the quality of cotton fiber before the cotton has been ginned. The system must be tested against standard quality measurements to establish a relationship between measurements made with the new system and standard measurements. To test this system a functional prototype must be built and presented with cotton in a controlled environment.

Design Considerations

A harvester-mounted seed-cotton fiber-quality sensor must be designed to meet certain physical constraints. Since the sensor would be attached to mobile agricultural equipment, physical size and durability with respect to its operating environment are limiting factors. There are also a number of required components that the system must have to accomplish its task of measuring, mapping, and recording cotton fiber quality.

Size Limitations

Sensor-mounting locations on a cotton harvester restrict the maximum width, height, and weight of the device, while components required to make the system work restrict the minimum limits. To sample freshly picked cotton, a sample-collection and sensor assembly should be mounted on a duct of a cotton harvester. The duct must be able to support the weight of the assembly and keep it stable while the harvester moves, or the duct must be appropriately reinforced. A duct has finite width and length available for mounting the assembly, and the assembly must not interfere with the duct as it telescopes from head movement, and it cannot obstruct the visibility of the operator. The focal length of the camera lens creates the minimum linear dimension between lens and sample window inside the assembly. If the lens is too close to the cotton sample, the image cannot be focused on the sensor. Once the distance is set from the cotton sample to the camera lens, the light sources must be positioned so they do not directly reflect

from the sample window onto the sensor. By keeping the physical size limitations in mind during the design, transition from laboratory testing to field work should be readily achievable, requiring few modifications to the laboratory prototype.

Environmental Factors

The sampling-and-sensing system must be durable enough to survive the environment in which it operates. When mounted on a harvester, the assembly will be subjected to frequent but unpredictable and sometimes strong vibrations, high temperatures, moist air and occasional precipitation, and dusty and possibly even muddy conditions.

There are two main sources of vibration that the system must be able to handle. The first is simply from the harvester's engine and fans that convey the cotton through the ducts, and the second is the flexing and jarring of the harvester as it moves through the field during harvest. Since the system's measurement of micronaire depends on the alignment of sequential images, the relative positions of a captured sample and the camera, lenses, sensor, and light source must stay constant. Thus the structure of the assembly must be rigid enough so that the vibrations and motion of the harvester do not cause the assembly to flex and change the position of the image on the sensor or the angle of the lighting.

Cotton is grown and harvested in warmer climates, and temperatures can sometimes be high during harvesting. The components in this system must be able to work continuously in ambient temperatures at least between 0°C and 40°C. Overheating during operation is the main temperature concern. The system will likely generate enough heat from the system's components to reach operating temperature in the cooler months. However, a temperature-regulated cooling system made up of, at the very least, a fan to circulate ambient air must be added before the system can be adapted for field application. For laboratory testing of a prototype, however, sufficient cooling is available from leaving the light source and other electrical and electronic components open to conditioned air between 20°C and 25°C.

Even though cotton is generally harvested when it is relatively dry, the system must be tolerant to moisture internal to the sampling device such that it can endure harvesting cotton when it is damp, and it must also be tolerant to external moisture such that it can withstand being exposed to rain. The electrical connections and the camera housing must be water tight while the system is not in use. The material from which the system is constructed cannot rust or otherwise corrode over time.

Because cotton is often harvested in dry and dusty environments, the system must prevent dust and dirt from accumulating on any of the optics or on the sampler window. If dust or dirt were to accumulate, they could affect the transmission of light through the system and negatively affect the measurement of cotton quality.

The design of the prototype seed-cotton fiber-quality system accounts for the structural requirements of the system such as height, width, and weight, as well as the power sources available on a harvester. The environmental conditions such as moisture, dust and heat are left to be considered later after the system has been tested and is ready to be adapted to in-situ cotton fiber-quality measurement.

Required Components

The following components are essential for the system to measure, store, and process images of cotton samples: (1) an image sensor, (2) a device to present cotton samples to the image sensor, (3) a frame that defines and maintains the relative position between image sensor and sample, (4) a lighting system to illuminate the presented sample, (5) optical filters and a mechanism capable of positioning and changing them to restrict reflected light to specified wavebands at appropriate times, and (6) a computer to process data collected by the image sensor and store it in non-volatile memory, and to serve as the controller for automated functions including changing filters and image analysis.

Prototype Design and Assembly

By using as many commercially available components as possible to fulfill these requirements, initial and maintenance costs of the system can be minimized. However, some components must be manufactured specifically for this application.

Hardware

Optical filters were selected for fiber-quality measurement based on wavelengths that have shown very strong correlation (R^2=0.99) to the HVI micronaire value of cotton fiber in the work of Sui et al., 2008: 1450, 1550, and 1600 nm. An image sensor that is sensitive in all the selected wavebands is needed. Thus, an NIR camera (Indigo Alpha NIR camera with a VisGaAs sensor, FLIR Technologies, Wilsonville, OR) was selected, and it is sensitive in the visible region as well, potentially enabling the use of visible reflectance to differentiate foreign matter from cotton fiber. Cotton "trash" -- leaf fragments, sticks, seed coat particles, and any other non-fiber material present in machine harvested cotton -- and cotton fiber have different reflectance properties throughout the visible and NIR portions of the spectrum (figure 1). A number of wavelengths within the sensitivity range of the camera have significantly different reflectance ratios (fiber reflectance over trash reflectance), so two, 650 and 1300 nm, were selected to discriminate between fiber and trash.

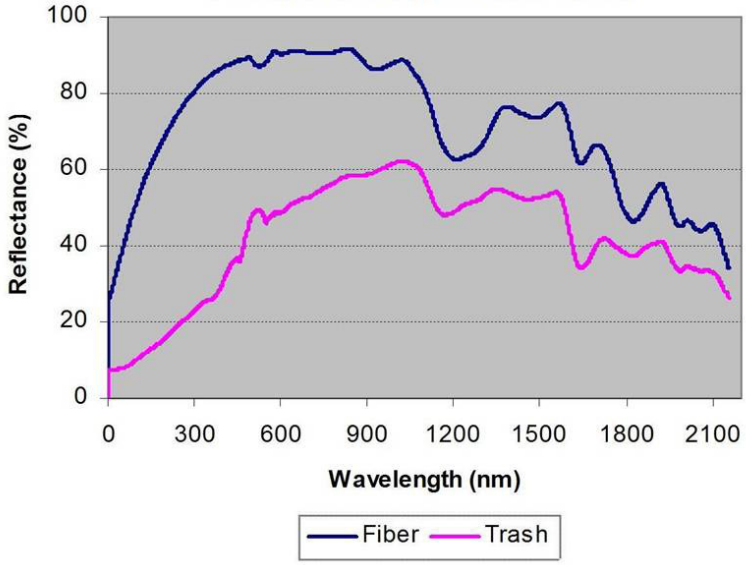


Figure 1: Spectral reflectance of cotton fiber and cotton trash (Thomasson and Sui, 2000).

The system requires the sensor to collect images of the same object through different optical filters. To accomplish this, the filters must be automatically changed without moving other system components. This can be done with a motor-driven optical filter wheel that has sites for filters related to the wavebands of interest. The motor that drives the wheel is controlled by the processor. The response time of the filter wheel will be a major factor in the speed at which data from each sample can be collected. Since changing the filter wheel position is a physical process instead of an electronic one, a faster response and activation time of the wheel means the current sample can be released faster in order for the next one to be collected.

Before an image can be collected from a sample, it is important to remove voids and shadows on surface objects. By pressing the cotton flat against a transparent window, illuminating energy can reflect from a roughly planar surface of cotton. The window must be made of a material that can transmit NIR energy as well as visible light and is strong enough not to crack under the stress of sample compression.

Once the image data from each sample have been collected and processed, the results must be stored until they can be read and analyzed. At the very minimum, the micronaire estimate and a sample identifier must be saved to non-volatile memory. During harvest the identifier would typically be GPS coordinates that indicate where the sample was taken. GPS data requires 8 bytes of storage space for each set of coordinates, and the micronaire value can be stored in 2 bytes. For minimal data storage, 10 bytes of disk space is required for each sample. If a sample were collected and measured every 30 s, the daily memory requirement would be less than 30 kB, a very reasonable amount. On the other hand, if the user desired to look at the image data later, each collected frame could be saved to non-volatile memory. As more data are required to be written to non-volatile memory, the amount of memory required increases. With the camera selected, each image has 256 x 318 pixels with 12-bit resolution, so storing five spectrally different images for each sample (assuming no image compression) would require a minimum of 3.1 MB per sample if values were stored as double precision (64-bit) binary arrays. (Double precision would be recognized by any typical platform that might be used for further processing.) It is unlikely that, in a commercial context, such a high memory requirement would be justifiable, particularly considering that cotton farmers would desire to have mainly a map of micronaire values and would not care much about the images.

The camera's sensitivity to both visible and near infrared energy allows visible and NIR images to be collected with the same sensor, making the overall sensor configuration simple (one detector) and thus making pixel comparison between images of different spectral bands easier. In such case, for a given sample, a pixel in different spectral-band images represents the same area on the cotton sample. With 12-bit resolution, each pixel has 4,096 (2^{12}) possible values. A typical 8-bit camera has only 256 (2^8) possible values for each pixel. Thus, a pixel in an image from the Alpha NIR camera can potentially provide a much more precise measurement of the reflectivity of the sample area that it represents.

A prototype of the sampling system was fabricated to collect data in a laboratory setting. The frame of the system is made from 25.4-mm (1.0-in.) square steel tube. Steel was selected for its durability and resistance to stress fractures from sustained vibration. If this system were implemented on a cotton harvester, it would be exposed to vibrations from the harvester operation and stresses from uneven terrain. Since the system compares separate images of the same object, any movement of the camera in relation to the cotton sample caused by the flexure of the frame would be unacceptable. Using a material (like adequately sized steel tubing) that has little susceptibility to fatigue over time helps ensure that alignment does not change as the system ages. The system was designed to be as compact as the optical components would allow so that harvester adaptation could be simplified. The minimum focal distance of the camera's lens determined the minimum height and width of the system, since the light source must be located outside of the image sensor's field of view as it is reflected off of the glass of the sampler window (figure 2).

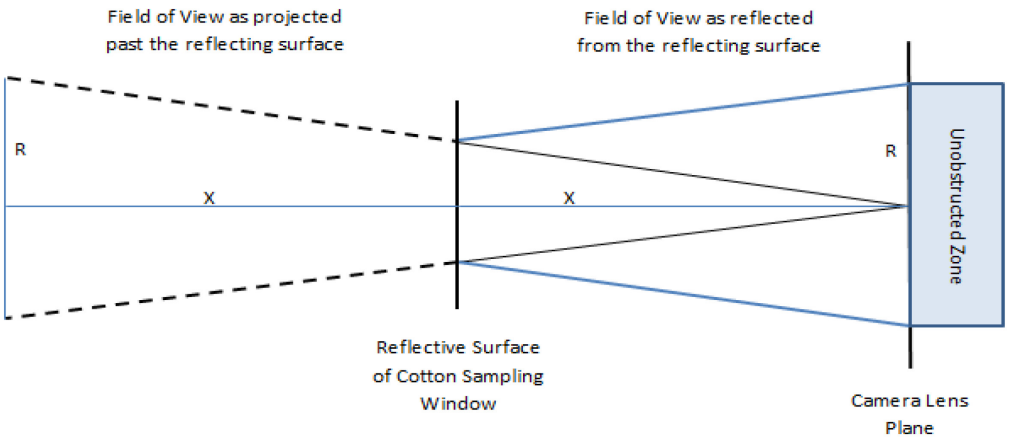


Figure 2: Illustration of the camera field of view as reflected off of the cotton sample window. X is the distance from camera to glass. R is the radius of the area centered on the camera lens that will be visible to the camera in the reflection at the surface of the sampler window and must remain unobstructed.

For purposes of developing the prototype in the laboratory, a Latitude D630 laptop computer (Dell Corp., Round Rock, TX) served as the processing and control unit. As opposed to using a dedicated processor like a single-board computer, this laptop PC facilitated sequential and iterative programming during system development. It also provided enough storage space for all the image data collected for experimental samples and a display for the operator to monitor while image data were being collected in the laboratory.

A PCI (peripheral component interconnect) "frame grabber" card (PCI-1422, National Instruments, Austin, TX) was used to connect the camera to the laptop PC (personal computer). This card allows use of National Instruments' NI-IMAQ library of programming tools, including high-level functions for image acquisition from, and serial communication with, the camera. Since PCI cards are not directly compatible the laptop PC used, an adapter from PCI to PCMCIA (Personal Computer Memory Card International Association) card format was needed, so a 1-slot PCI expansion system (MAGMA, San Diego, CA) was used.

To acquire images in wavebands centered at particular wavelengths, a set of optical band-pass filters were selected for their transmission properties in the desired wavebands: part numbers FB650-40 (figure 3), FB1300-30 (figure 4), FB1450-12 (figure 5), FB1550-12 (figure 6), FB1600-12 (figure 7), all from ThorLabs (Newton, NJ).

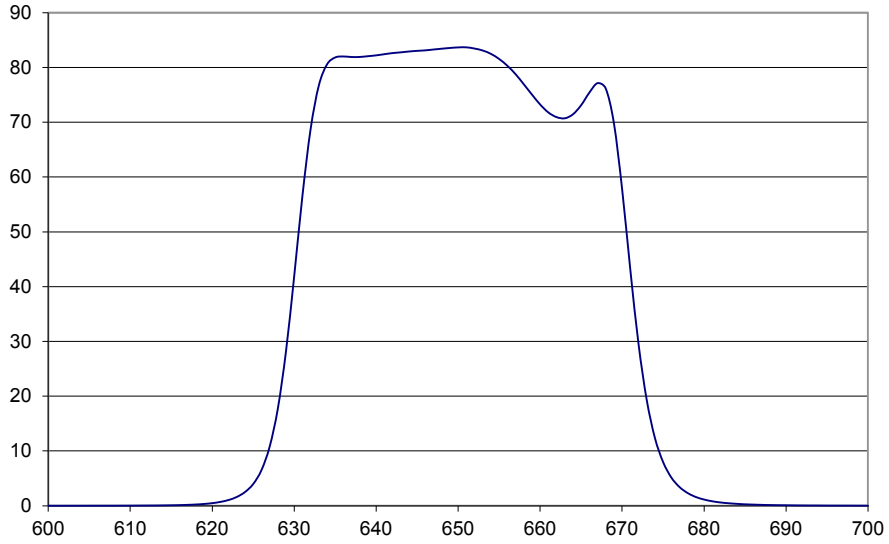


Figure 3: Transmission curve for the 650-nm band-pass filter with 40-nm FWHM (full width at half mean).

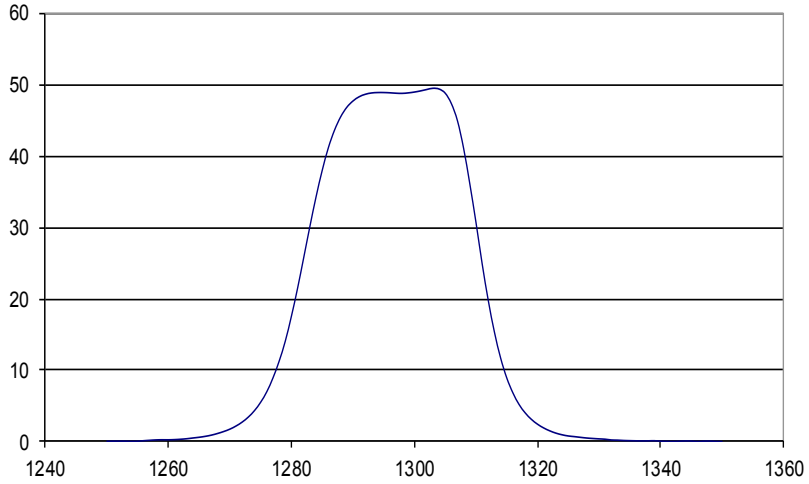


Figure 4: Transmission curve for the 1300-nm band-pass filter with 30-nm FWHM.

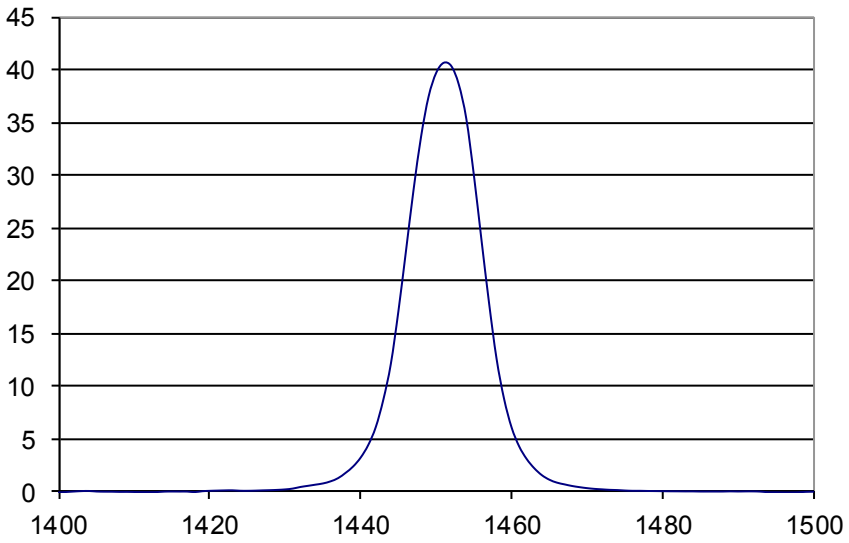


Figure 5: Transmission curve for the 1450-nm band-pass filter with 12-nm FWHM.

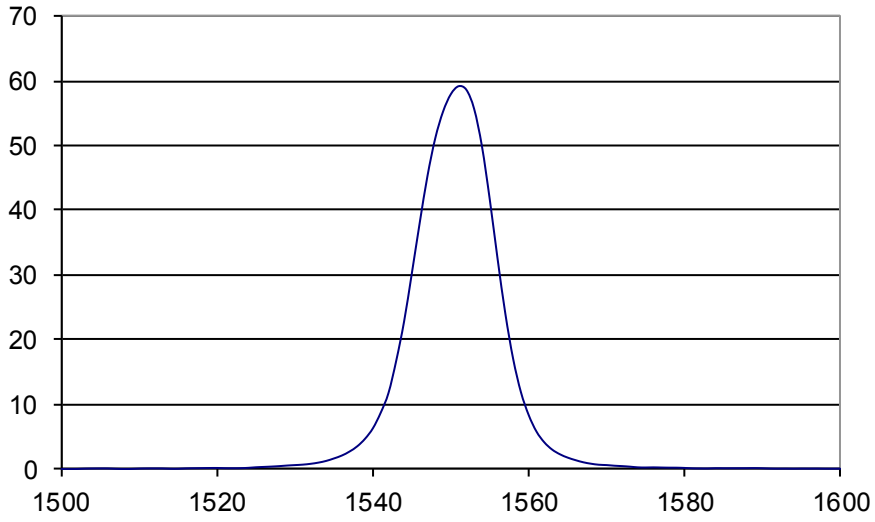


Figure 6: Transmission curve for the 1550-nm band-pass filter with 12-nm FWHM.

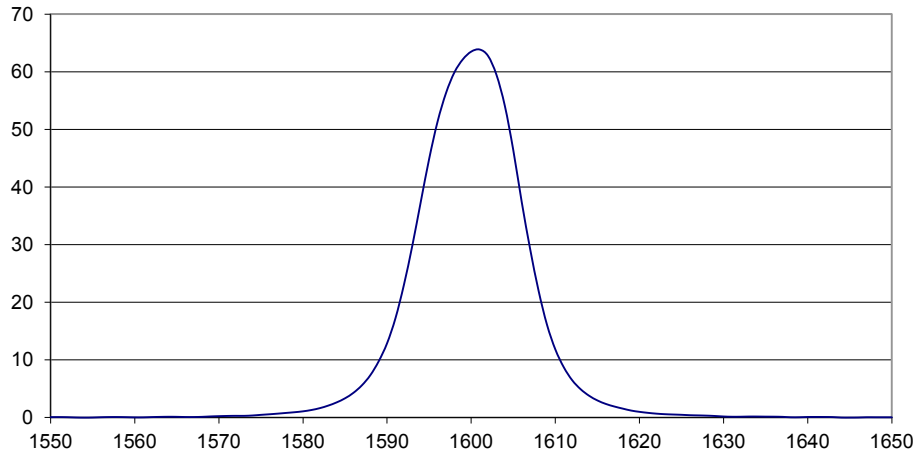


Figure 7: Transmission curve for the 1600-nm band-pass filter with 12-nm FWHM.

When these filters' spectral transmission properties were combined with the spectral properties of light-source emission and camera-detector sensitivity, a large amount of variation existed in the amount of energy available to the sensor through the different optical filters. Since the camera lens used did not have electronically controlled iris that could be used to reduce available energy in certain wavebands as needed, neutral density (ND) filters were added to the optical filters as necessary to balance the energy detected by the camera through each spectral filter for the expected reflectance of cotton. The attenuation necessary for each ND filter was found by using spectral data for each component interacting with the light energy as it travels from the source to the sensor. This path, starting with the QTH lamps, travels through the sample window, reflects off of the cotton sample, travels through the sample window again, enters the optics through the 25-mm camera lens, through the first bi-convex lens, through the selected band-pass filter in the filter wheel, then past the second bi-convex lens, and finally falls on the sensor. Some of these components are "spectrally flat" across the spectral region of interest and can thus be neglected. For example, the bi-convex lenses transmit 99% of available light from 400 to 2,000 nm and were thus

excluded from the energy-balance calculations. The components that are not spectrally flat are the QTH lamps (figure 8), the reflectance of the cotton sample itself (figure 9), each of the band-pass filters, and the imaging sensor (figure 10). The relative level of energy detected by the sensor as a function of wavelength was found by taking the product of the relative intensity of energy provided by the QTH lamps and the interaction with each of the optical components on the path to the sensor. This remaining relative energy level was multiplied by the relative sensitivity of the camera's VisGaAs detector (figure 11), and the area under each curve was calculated to provide the level of camera sensitivity for each filter. Fine correction necessary to further equalize the energy level in each spectrally different image was handled in software by modifying the sensor integration time of the camera.

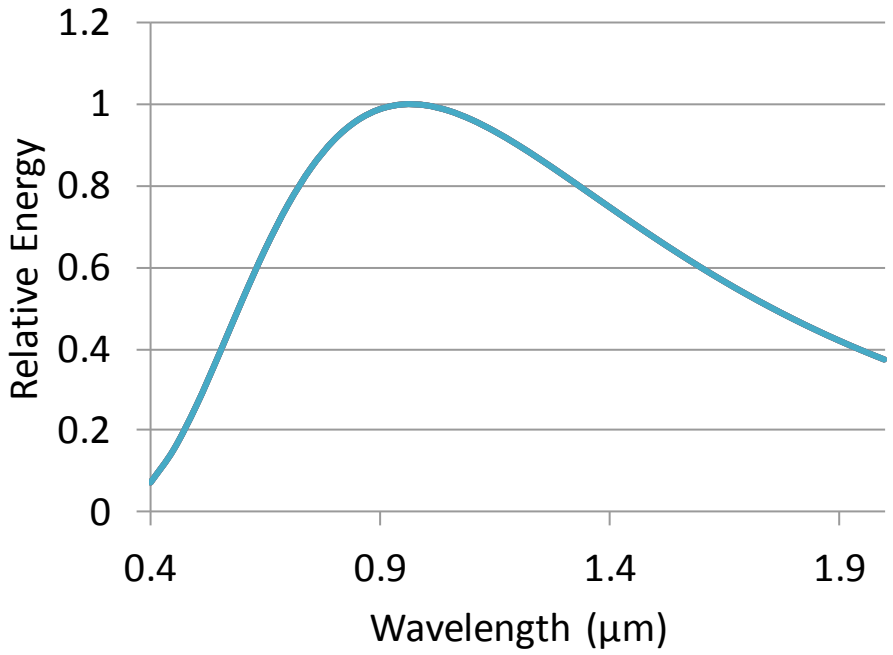


Figure 8: Spectral data for the relative energy provided by the QTH lamps.

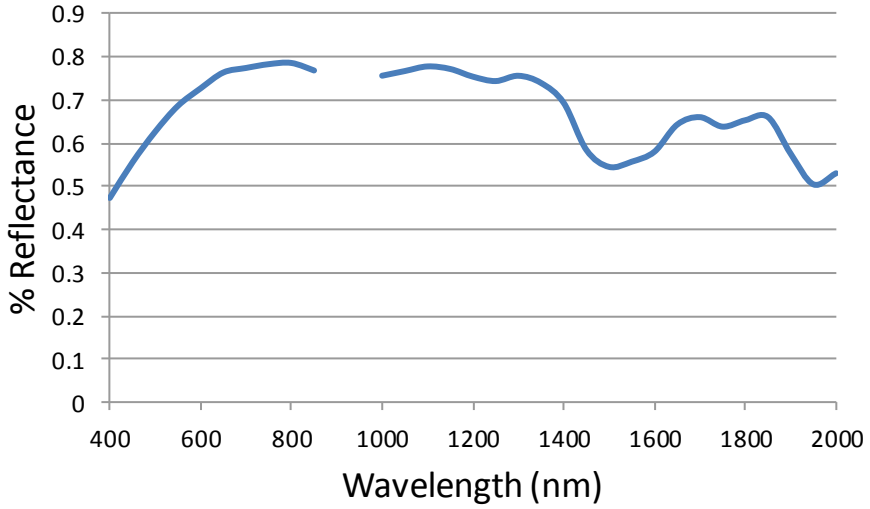


Figure 9: Average spectral reflectance data for cotton fiber (Sui et al. 2008).

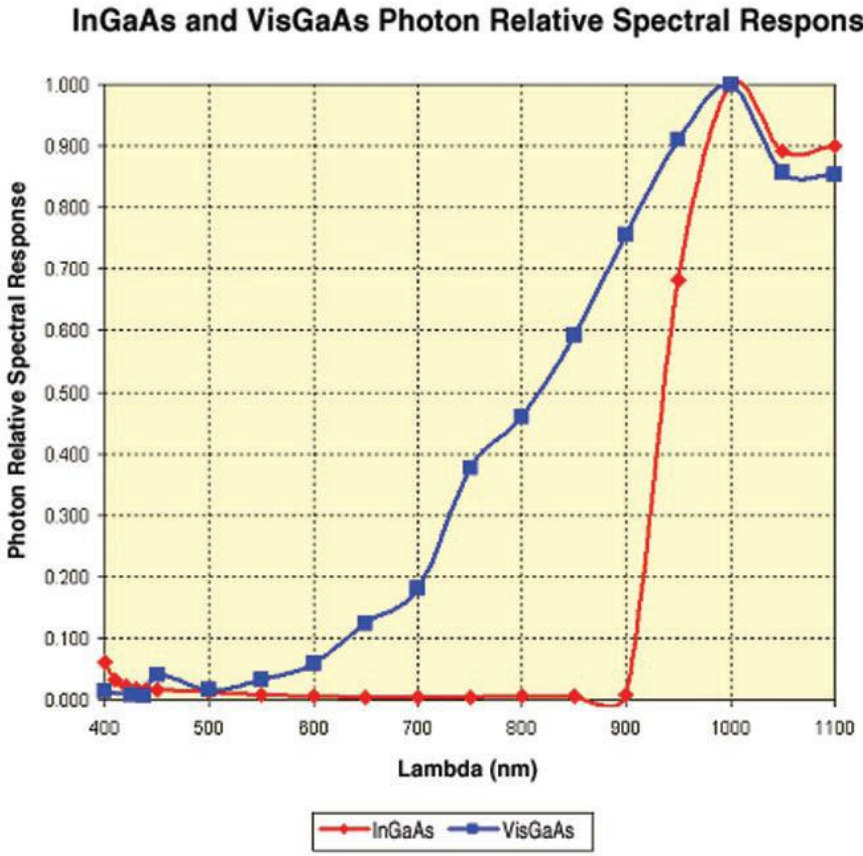


Figure 10: InGaAs and VisGaAs Photon Relative Spectral Response (Walker, 2004).

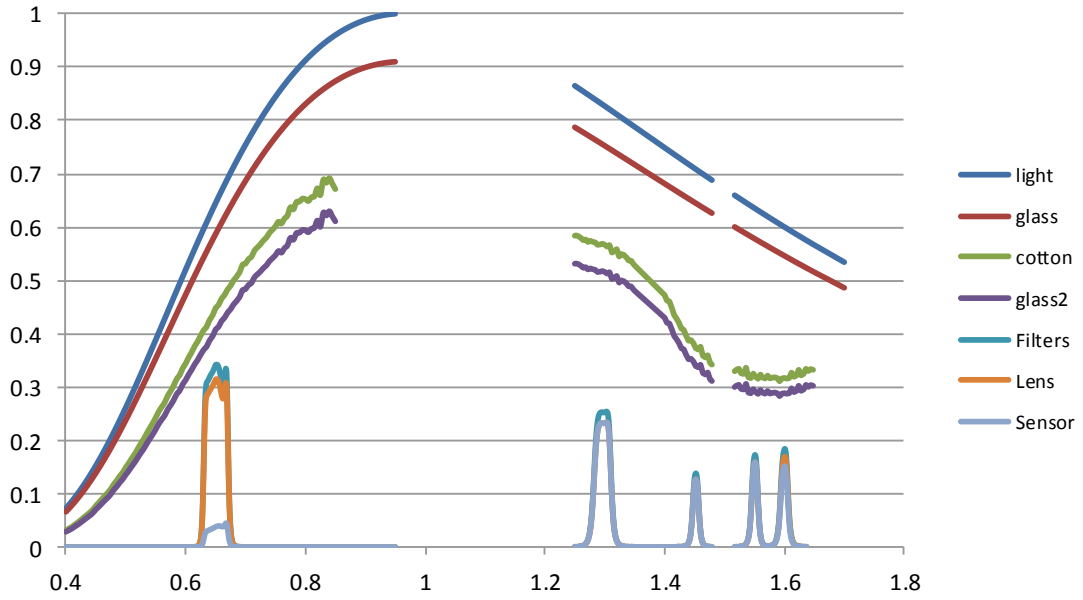


Figure 11: Energy attenuation graph. Displays the loss of energy as light passes through each optical component.

A motorized optical filter wheel (model F102B, ThorLabs, Newton, NJ) was selected for the purpose of changing optical filters during operation. This filter wheel can be directly mounted to the camera, between the camera body and the lens, ensuring that all light energy reaching the sensor has gone through the selected filter. The filter wheel allows the filters to be quickly and automatically changed without opening or moving any other part of the optical system. The motor is controlled by way of a USB-to-serial adapter, allowing the filter wheel to connect to a USB port on the PC and be controlled with serial commands.

A series of lenses must be used to focus the image onto the sensor. In a common application the c-mount camera lens would be attached directly to the camera. The focal point of this lens would fall directly onto the camera's detector, and no additional optics would be required. However, since mounting the filter wheel between the camera lens and the image sensor added distance between the two, a set of bi-convex lenses was needed to transmit the image across the filter wheel (figure 12). The first lens uses the

image provided by the camera lens as its object and thus must share a focal point with the camera lens. The image of this first lens is focused to infinite distance through the filter wheel. The second bi-convex lens is focused to infinite distance for the object to be imaged. The focal point of this second lens falls on the image sensor in the camera body. This sequence takes the light energy that was focused to infinity by the first bi-convex lens through the filter wheel and focuses it onto the image sensor. Using this lens setup to relay the image across an optical filter will minimize distortion if the image is transmitted through filters of different thickness, because light energy focused to infinite distance passes through the filter perpendicular to the direction of travel and will thus not be refracted at all.

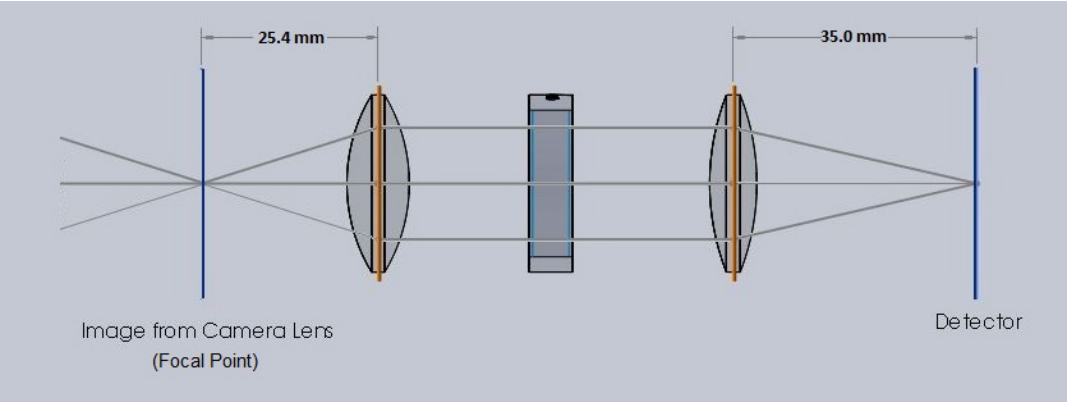


Figure 12: Sketch of the bi-convex lens setup to transmit the image across the filter wheel.

A pair of 50-W quartz-tungsten-halogen (QTH) lamps (MR16 Superline™ Reflecto™ Series, Ushio Corp., Cypress, CA) were selected to serve as the light source because of the consistent spectral output QTH lamps provide over their life. Using QTH means that the system needs time to reach a consistent operating temperature, as the output varies with the temperature of the bulbs, and the temperature takes some time to stabilize. One of the most important factors in selecting the light source was that it must be able to be powered by a 12 VDC system, like the power source found on a cotton

harvester. The lamps selected have an advertised color temperature of 3000K. They were selected also for their coating, which reduces back spill of infrared energy by 80%. This feature reduces the temperature of the lamp sockets and increases the amount of NIR light emitted toward the cotton sample. They also utilize an axial filament and a "multi-lens geometry" on the reflector surface, providing a relatively even light distribution, free from shadows and bright spots. The lamps are placed in diagonally opposed upper corners of the frame. The corners were chosen to allow the lamps to be set at the greatest possible distance from the vertical centerline of the sampler. If the lamps are too close to the centerline, there will be a direct reflection of light from the source to the sensor. Any light from the QTH lamps directly reflected from the sampler glass to the sensor would not be representative of the underlying cotton, thus interfering with measurement of cotton reflectance.

For purposes of the laboratory prototype, the sampling window (figure 13) is integrated with the sampling press. The assembly is made of two 6.3-mm (0.25-in.) thick PVC sheets that measure 310 x 230 mm (12.2 x 9.1 in). Holes were drilled 25.4 x 25.4 mm (1.0 x 1.0 in.) from each corner of each sheet (eight in all) to allow four 9.5-mm (0.375-in.) carriage bolts to be inserted to hold the sheets together. Into one of the sheets (the "top" sheet) a 150 x 150 mm (5.9 x 5.9 in.) hole was cut, and a 152 x 152 mm (6.0 x 6.0 in.) piece of 3-mm (0.12-in.) thick Borofloat glass plate was glued to the bottom side. A thickness of 3-mm was selected to give the glass enough strength to withstand the pressure exerted onto the cotton samples. This glass transmits 91% of light in the 400-nm to 2000-nm range.

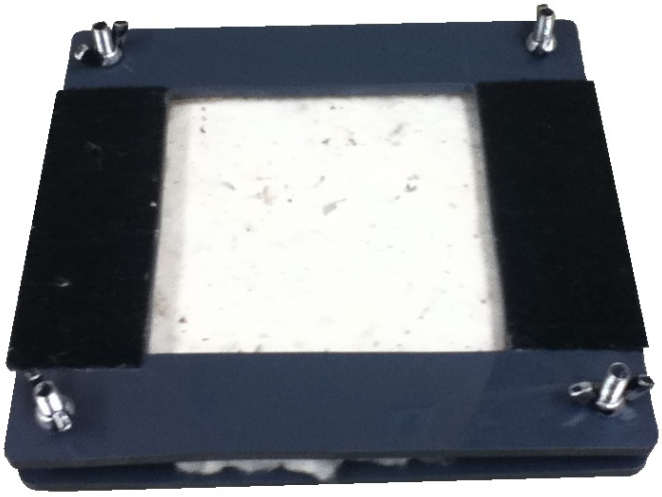


Figure 13: Cotton sample imaging press and window.

Software

Software was developed to control operation of the prototype and for data analysis. Three principal algorithms were required: a system-control algorithm, a calibration algorithm, and a data-processing algorithm. The system control and calibration algorithms (Appendix A) were written with the C++ programming language, while the data processing algorithm (Appendix B) was written partly in C++, for the built-in processing algorithm, and partly in MATLAB, for processing done after image collection.

During automated operation as is expected on-board a harvester, the system must take an input from the sampling mechanism that indicates a cotton sample has been collected and is positioned in the sample window, and then it must direct the filter wheel to position the appropriate optical filter between the image sensor and the cotton sample. Once the filter is in place, the processor must read the image data from the sensor and store it before it directs the filter wheel to move again. When image data have been collected through each optical filter, the processor must direct the sampling mechanism to release the current sample and begin collecting the next. While the next sample is being collected, the processor uses the images collected to identify foreign material and

then measure the reflectance value of the cotton lint. In the current laboratory prototype stage, the system is not automated, so the sampler input must be given by the operator. When the system is adapted for harvester-based operation, an automated sample collection mechanism will be incorporated. In field applications each sample will be identified by GPS coordinates to allow the resulting micronaire measurement to be mapped to a unique point in the field.

Regular calibration is important for any sensing system. The calibration program developed for the prototype sensor adjusts for variation in illumination on the cotton sample across the spectral region of interest. After the QTH lamps are turned on and allowed to reach operating temperature, a plain "white" reference (e.g., paper or ceramic tile) is placed into the cotton sample holder and presented to the imaging sensor as a cotton sample would be. The system acquires a set of images and creates a pixel offset map to correct for non-uniform illumination (NUI) for each of the five optical filters. This map is created by taking an average illumination value from a 21 x 21 pixel area in the center of the image, then calculating a ratio for each pixel relative to that average value. During actual data collection, this map of pixel-illumination ratios is applied to the data supplied by the camera by multiplying each pixel value by its respective ratio.

Completion of the laboratory prototype required adequate hardware and software configuration, to demonstrate that the prototype system is capable of processing data during sampling. Thus, a basic image-processing algorithm was included with the system control algorithm. The process used during sampling involved (1) calculating a ratio of the 650-nm and 1300-nm waveband images, and then (2) classifying any pixels in the resulting ratio image as "cotton" or "not cotton". Threshold limits were used as the means of classification. Both high and low threshold limits on pixel values were calculated, and they were based on a histogram of the pixel values in the image of the ratio cotton sample reflectance in the two wavebands, and then to use those limits to classify the pixels in the image as either "cotton" (between the threshold limits) or "non-cotton" (outside the threshold limits). A threshold of 1.308 to 1.355 was used; between those values pixels were identified as "cotton" by giving them a logical value of "true"

(bit value: 1) in the resulting binary mask image. Pixels having a value outside of these limits were given a logical value of "false" (bit value: 0) in the mask image to indicate that they represent pixels of something other than cotton fiber. This threshold range was chosen by observing histograms of the 650-nm to 1300-nm ratio images collected from five randomly selected samples, and then selecting upper and lower thresholds that distinguished between "cotton" and "non-cotton" in each sample. Once the mask image is complete, the average reflectivity of the 1450-nm, 1550-nm, and 1600-nm images is calculated. To do this while excluding "non-cotton" pixels, each NIR image is observed in relation to the mask image, pixel-by-pixel. If the pixel in the binary mask image is "true", the value in the NIR image is added to a cumulative sum, and a pixel counter is incremented by one. After every pixel has been considered, the sum is divided by the counter to reach the average value of pixels representing cotton fiber in each of the three NIR images.

Experiment

A laboratory-based experiment was conducted to evaluate the prototype seed-cotton fiber-quality sensing system, to demonstrate that image-based trash removal works in seed cotton, and to show that a relationship exists between seed-cotton fiber reflectance and HVI micronaire values. To demonstrate the field utility of the system, seed cotton samples were collected during a normal machine harvest.

Sample Collection

A long term goal of this project is to incorporate the prospective sensor into a harvester to measure fiber quality of machine-harvested seed-cotton samples. To ensure the results of prototype testing are applicable to a real-world situation, it is necessary to use machine-harvested seed-cotton samples during testing. Gathering machine-harvested cotton samples is important because of the effect that machine harvesting has on fiber properties. For example, different harvesting methods and ginning techniques can cause variation in the micronaire value of cotton fiber (Calhoun et al. 1996).

Cotton samples were collected from fields at the IMPACT Center of the Texas AgriLife Research farm in Burleson County, Texas. Field sample locations were selected within the cotton field to be harvested. Research by Stanislav and Morgan (2007) indicated a relationship between apparent soil electric conductivity (EC_a) and the micronaire value of cotton fiber, so EC_a data from each of the fields were used to locate sampling points. The EC_a data were split into three groups (low, medium, and high), and stratified random sampling was used to generate 12 sampling points in each EC_a group in 2008 (36 points total) and 16 points in each EC_a group in 2009 (48 points total). This process was intended to increase the variation in micronaire values among samples. A handheld GPS receiver was used to place a marker flag at each of the points prior to harvest.

At each of the flagged points, machine-harvested cotton samples were collected from the duct of a John Deere 9965 cotton picker. When a flag was reached, the harvester was stopped, the flag was removed, and a sample was collected immediately after the harvester resumed operation and placed into a numbered paper bag. The samples were collected with a gloved hand placed in the airflow at the duct outlet in order to intercept the cotton before it went into the picker basket. Each handful was placed into the paper bag, and the process repeated until the bag was at least half full. Collecting the samples in this way ensured that at least 0.25 kg was collected, guaranteeing that the cotton could adequately cover the sample window of the prototype instrument. Each sample collection took between three and six handfuls of seed cotton depending on the harvester's momentary mass flow rate.

Data Collection

Before each data collection period the prototype system was turned on for warm-up and pre-operation checks. The QTH lamps were allowed to warm up for 30 minutes to reach a constant operating temperature prior to data collection. During this time, a check of alignment between the motorized filter wheel and camera was performed. The filter wheel was cycled through all five positions to verify that it was operating properly,

and alignment between the camera and sample window was checked. A set of images was acquired with the camera to ensure that there was sufficient delay to allow a filter change before the next image was recorded. Positioning marks were made relative to the prototype frame so that proper placement of the sample window could be repeated for each sample.

To ensure the uniformity of the images taken, the system had to be calibrated before use. Since non-uniform illumination can be a significant problem when dealing with image processing, flat-field correction was performed as described previously. This prototype was used in a controlled-environment laboratory setting over a short time window. In a production sensing system, provisions must be made to allow adjustment of gain and offset during system calibration. Without such adjustment, over time, measurements made by the system could not be related to past measurements.

A 100-g (0.22-lb) subsample from each of the 86 seed cotton samples (36 from 2008 and 48 from 2009) was presented to the imaging system. The subsample was placed into the sample press and pressure applied by evenly tightening the wing-nuts until the voids between cotton bolls and around any surface objects collapsed. The sample press was then placed before the camera, aligned with the positioning marks, and the image collection was then started. To initialize data collection at the beginning of a data collection session, the user must enter the number of the serial communications port used by the filter wheel and the drive letter on which to store image data. To start data collection, the user must enter an ID for the current sample and press “Enter” on the keyboard. The computer display shows the progress of image collection and indicates when image collection is finished. Afterward, the sub-sample is removed from the press and returned to its original sample bag so the next sample can be prepared for data collection. While the next sample is being prepared, the system processes and saves the images it has just collected. To account for variability within samples, three replications were collected from every eighth sample. Between replications, each sub-sample was replaced and mixed back into its original sample so that another sub-sample could be collected randomly later.

To determine whether the image-based method of trash removal was effective, it was necessary to compare the reflectivity data between the seed cotton samples and lint samples, in which most of the trash had been physically removed. Once imaging of the machine-harvested seed cotton was complete, the seed cotton samples were ginned with a saw gin at the Cotton Improvement Laboratory at Texas A&M University in College Station, Texas. Each sample was ginned separately and the cotton lint placed back into its respective storage bag. The lint cotton was then taken back into the laboratory, and the image collection process was repeated with the ginned lint cotton samples. A 100-g (0.22-lb) subsample of the cotton fiber was placed into the sample press, and pressure was applied until the sample window was free of voids and shadows in the cotton fiber. The sample press was presented to the imaging system, and the user initialized the system and started the image collection process. The lint cotton samples from the 2009 harvest were replicated in a similar way to that of the seed cotton samples. The samples from the 2008 harvest were not replicated because there was an insufficient amount of cotton fiber to create subsamples after the ginning process. After images of each subsample were collected, a 20 to 25-g (0.044 to 0.055-lb) portion of cotton fiber was placed into a labeled plastic bag and sent to the Fiber & Bio-Polymer Research Institute in Lubbock, Texas for HVI analysis.

The images of the cotton samples were stored to non-volatile flash memory along with the binary mask for each sample, and a summary of the calculated values. To make sure the image data could be easily read into and used by different image processing programs being considered for further analysis, they were saved in two different formats: as a binary file in the form of a double precision array, and in a delimited text file. While the data for each 12-bit pixel could fit into a “half” (16-bit) or “single” (32-bit) precision array, “double” precision (64-bit) was chosen for its widespread use in image processing and analysis programs. The delimited text file is readable when opened with a text editor, and can easily be imported into spreadsheets. The data in this type of file are 7-digit numbers that represent their corresponding pixel values (0000.000 –

4095.000). Each pixel is set off from other pixels in its row by a space, and each row is set off by a new-line character.

The images from each sample were also saved as Portable Network Graphic (PNG) files. This format was chosen because it does not use compression to reduce the file size. Compression algorithms would compromise the data and reduce the amount of information available. By using a standard image format, the image can easily be displayed, allowing the user to inspect the images to make sure they do not show any unexpected occurrences; for instance, an image taken while the filter wheel is between positions.

The mask file was saved to allow the user to identify which pixels in the images were used in calculating the reflectance of the cotton fiber during sample collection. This file can be loaded into an image-processing program to be viewed or used in further processing. It consists of strings of ones and zeros, each of which represents a pixel. Each row is separated by a new-line character, and there is no separation between pixels within a row.

A summary of the results for each sample is saved as a text file. This file includes the ID of the sample, the calculated mean pixel value for the 1450-nm, 1550-nm, and 1600-nm images, a count of pixels used in the mean calculation, and a short description of the method used to create the image mask. This file allows the user to identify images that might have a low pixel count, or mean values that don't follow the trend of the rest of the samples.

The micronaire value measured for each sample was appended to the reflectance data, and multiple-linear regression was conducted with each of the selected trash-removal image-processing methods to find the one that produced the strongest correlation between image-based data and HVI micronaire.

Image Data Analysis

During sampling a simple pixel classification process based on the histogram method used by Sui et al. (2008) was applied to the images of the cotton samples. This

process was written into the control algorithm in order to demonstrate the ability of the system to process the images immediately after acquisition and calculate an estimated micronaire value while of the next seed cotton sample is being positioned for imaging.

The cotton fiber calibration samples used by Sui et al. (2008) were used as control samples in this system. These samples are very uniform, and basically free of trash, so there is very little to mask out. While reviewing the data, the stored mask files were loaded into MATLAB to visually inspect what the algorithm was masking out in order to find trends in the mask images. For instance, with a lighting uniformity problem the same portions of each sample might be masked out due to a lower reflectance value simply because there is a lower amount of energy incident on those portions (figure 14). This non-uniform lighting issue was corrected using flat-field correction. Another way the data were initially reviewed was by inspecting the summary file created for each sample during data collection. This review considered the number of pixels that the method used to calculate the average reflectance of each of the NIR images; i.e., the number of pixels that were classified as "good" out of the total of 81,408.

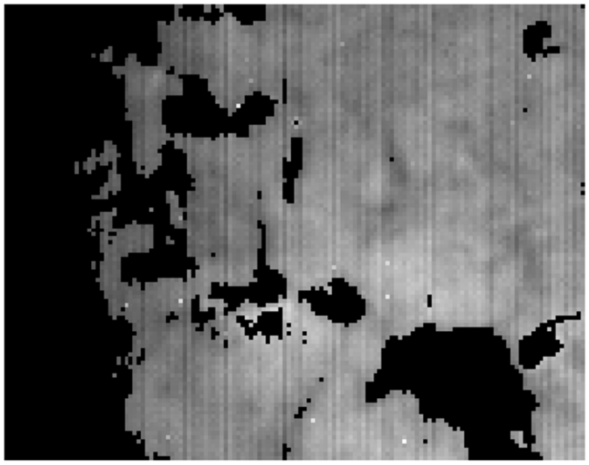


Figure 14: Masked image showing NUI problems. The left side of this image had a lower reflectance simply because there was less lighting.

Images were processed after all sample images were collected in order to determine the best method to remove trash effects and estimate the micronaire value of each cotton sample. This image processing was done with a script written for the MATLAB program (Appendix B). This script uses the images from each sample to identify pixels in the NIR images that represent cotton fiber, then solves for the average reflectance of cotton lint in each of the NIR images. The script finally adds the HVI measured micronaire value for the cotton sample and generates a file with the sample ID, three reflectance values, and the HVI micronaire value.

Two methods were considered for analysis with scripts written in MATLAB. The first, a dynamic threshold method, is an automated version of a method similar to the histogram method used by Sui et al. (2008). The other, object detection, uses established image processing techniques to find objects within an expected background of cotton fiber.

The dynamic threshold method uses an image histogram to find a range of pixels that most likely represents cotton fiber based on frequency of value occurrence. This method was manually used by Sui et al. (2008) and resulted in a strong relationship between cotton fiber reflectance in certain NIR wavebands and the cotton's HVI micronaire value. This method of pixel classification is similar to the processing method built into the system control software. The same basic steps are required, but with dynamic threshold limits the values for the threshold limits are calculated anew for each cotton sample and vary because they are based on properties of the pixel-value distribution in the image.

Within dynamic threshold limits, three procedures for establishing classification limits were explored. The first is a procedure centered on the mode (mode procedure) of the pixel-value distribution. This procedure sets the threshold boundaries based on their distance from the most frequently occurring value in the distribution. The second, a procedure centered on the median (median procedure) of the pixel distribution, sets the threshold boundaries based on their distance from the value(s) that fall(s) in the exact center of the sorted distribution. The third is a procedure that is strictly based on

excluding a certain pre-set percentage (percentage method) of pixels from each end of the distribution.

The mode procedure was applied to an image ratio of the 650-nm and 1300-nm images as well as to the individual images themselves. To obtain the image ratio, the pixel values in the 650-nm image are first divided by corresponding pixel values in the 1300-nm image, creating a ratio image. The pixel values in the ratio image are then plotted in a histogram, and the mode is found. The histogram is then used to find a lower boundary that excludes at least 10% of the 81,408 available pixels by adding the pixel count in each histogram bin to a cumulative sum until that sum is greater than or equal to 8,141. Values within each histogram bin cannot be distinguished any further, which prohibits the exclusion of the exact percentage desired. The pixel-value distance between the mode and the lower boundary (x) is calculated, and an upper boundary is created by adding 150% of x to the mode value (figure 15). If a single image is used to find the classification boundaries, the upper boundary is found by excluding 12% of the pixels from the higher end of the histogram, then setting the lower boundary an equal distance from the distribution mode on the opposite side (figure 16). These boundaries are used to classify the pixels as either “cotton” or “other”. Based on these classifications, a binary mask image is created in which pixels classified as “cotton” are given a logical value of “true”, and other pixels are given a logical value of “false”. The mask image is then used with the 1450, 1550, and 1600 nm images to find an average reflectivity value for the cotton lint by only using those pixels which have a logical value of “true” in the mask image to calculate the average reflectivity.

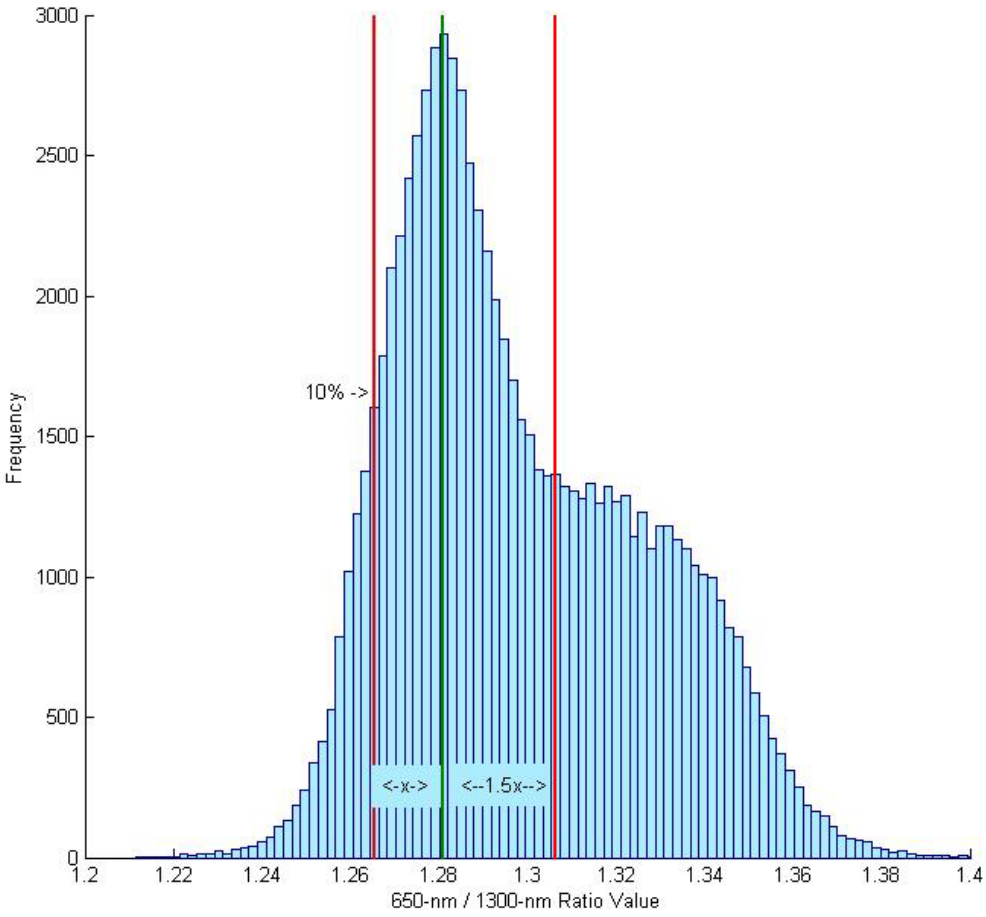


Figure 15: Cotton fiber pixel classification boundaries created by using the mode method on a histogram of a ratio of images.

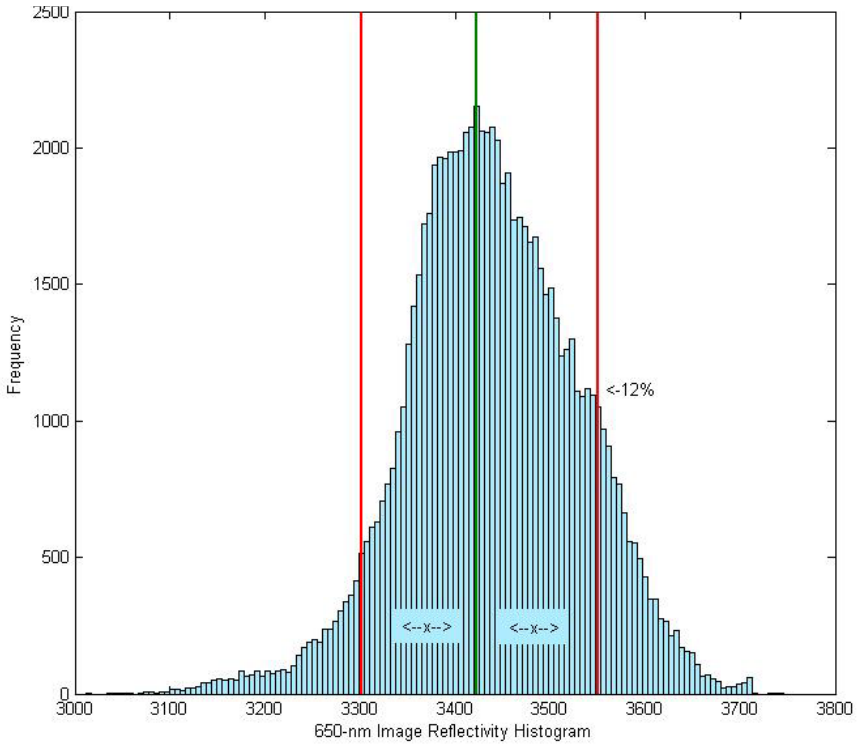


Figure 16: 650-nm histogram showing the "cotton fiber" boundaries (red) and the distribution mode (green)

With the median procedure, a histogram is again generated from the ratio image or a single image. Based on the histogram, the median and mode are calculated. The procedure then branches to one of two paths. The first path is taken when the mode value is less than the median value. This is the expected shape of a histogram from an image or a ratio of images at the aforementioned wavelengths for the average cotton sample. In this case, the histogram is used to make an upper boundary that eliminates 12% of the 81,408 pixels available. The lower boundary is then created by doubling the pixel-value distance between the upper boundary and the median value. The second path is taken if the mode value is greater than the median value. This situation could occur if the cotton sample contains a very high percentage of foreign material. In this case, the upper boundary is found in the same way as with the first case, but in this case the median is used as the lower boundary. For both cases, a binary mask image is

created in which any pixel that lies within the boundaries (inclusive of the boundary values themselves) is given a value of "true", while the rest are given a value of "false." The mask image is then applied to the images in the 1450-nm, 1550-nm, and 1600-nm wavebands. An average is then found for each of these three images based on only the pixels that correspond to a value of "true."

The percentage procedure also uses either a ratio image or a single image. A histogram is generated and used to keep 60% of the available 81,408 pixels. An upper boundary is set to exclude 12% of the pixels, and a lower boundary is set to exclude 28% of the pixels. The unequal pixel exclusion on the high- and low-value sides of the histogram is due to a lower observed pixel value of trash in the image ratio and the individual images. The percentage of excluded pixels must reflect this fact. The remaining 60% is considered to be cotton. A mask image is created with a value of "true" for any pixel in the 60%, and a "false" value for the other 40%. This mask is combined with 1450-nm, 1550-nm, and 1600-nm images to find an average reflectivity value for the cotton fiber in each image.

Each of these three procedures is a different way to reach the same goal, to set the pixel classification threshold boundaries based on the distribution of pixel values. All the percentages and multipliers were found based on a combination of considering histogram shapes and trial-and-error efforts with different values. Because these procedures tended to have similar results, and due to time constraints, only the mode procedure was programmed into MATLAB. If the distribution of pixel values in a sample image has a large standard deviation, this procedure appears to be the best at accounting for that while setting the boundaries when the boundaries are found manually. The other method considered of classifying pixels within the images was object detection. Object detection in image processing can be used to find contiguous sets of pixels in an image that do not have the same properties as the background of the image. In this case, it is expected that the image background is cotton fiber that will have pieces of seed, stick, bark, and other non-fiber objects that are picked up by a cotton harvester. Object detection can classify pixels as either "background" or "object"

by using various techniques to locate the edges of the object (edge detection) and then determining its boundaries. Edge detection is a method of locating points in an image where there is a distinct contrast from one area of the image to the next. When many of these points are found together it can be identified as an "edge". Once all of the edges are found in a single image, they can be used to find boundaries and identify objects. Edge detection works better on high-contrast images.

Thresholding based on a moving average is a form of edge detection that accounts for variation in the background from one part of an image to another. A starting value for the moving average must first be established. It can either be known or assumed beforehand, or found with random sampling in the image to set an initial average. The image is then examined pixel by pixel, comparing each pixel value to the established average multiplied by a thresholding factor. If the pixel has properties within threshold limits of the moving average, the value is included in a new moving-average calculation and classified as "background". If the pixel is outside the threshold limits of the moving average, it is classified as "object" and excluded from the moving-average calculation.

The dynamic threshold limit method with the mode procedure was selected for use in further analysis. This method was chosen based on (1) similarity to research that had already been completed, showing a strong relationship between NIR reflectance of cotton fiber and micronaire (Sui et al. 2008); (2) good flexibility to variations in apparent trash content when manually applied to image histograms; and (3) simplicity relative to edge detection. This method and procedure were applied to either a ratio of two images or single images. The histogram of a single image tended to have the reverse appearance of the ratio histogram. Cotton fiber is expected to make up a majority of the image, and thus associated pixel values are closer to the mode than trash pixel values. However, trash content varies from sample to sample, and it is possible for a sample to be made up of more trash that cotton fiber. In this rare event, the distribution of pixel values is reversed; more pixels in the images represent trash than cotton fiber. In this case the upper boundary (the boundary on the "short side" of the distribution), is set by excluding

a preset number of pixels instead of the lower boundary. The ratio of boundary distances to the mode work in similar manner to the more common distribution; the difference of the first set boundary and the mode are taken, then that value is multiplied by a pre-set ratio and subtracted from the mode value.

Both the single-images and ratios were used in multiple trials with the MATLAB functions (Appendix B). The input parameters used to establish histogram boundaries for the cotton fiber threshold are altered by way of a nested "for" loop to find the optimum parameters. The initial trial used parameters that excluded 5% of the pixels to set the first limit and set the second limit with a 1:1 ratio of distances from the mode. The percentage excluded was then changed in 1% increments for subsequent trials until a peak in the correlation was observed. Subsequently, while holding the percentage of pixels excluded constant, the distance ratio for the second boundary was varied by 10% from 1:0.5 to 1:3 to find the best ratio value for each method.

Statistical Analysis of Prototype Output

Once data collection and processing were complete, and the results from the HVI analysis of the samples were received, the two data sets were used to find a relationship between the average reflectivity of the masked NIR images and the measured micronaire value. Linear regression was used to establish a relationship between the average value of unmasked pixels in each image with each of the three NIR wavebands (1450, 1550, and 1600 nm) as independent variables and the measured micronaire value as the dependent variable. The relationship was established for every iteration of the processing method to find the model with the best fit by maximizing the r-squared value. Once the model with the best fit was found for both the single-image method and the ratio-image method, the residuals were observed for normality and homoskedasticity.

RESULTS

Design

The design of the seed-cotton fiber-quality sensor incorporated the research of Sui et al. (2008), which identified fiber-quality sensitive wavelengths that could be used to optically estimate micronaire. The sensor concept involves using intensity images of seed cotton in these wavelengths to make real-time in-situ cotton fiber quality measurements during a cotton harvest. Ultimately, these quality measurements can be combined with GPS coordinates to create high-resolution quality maps of a cotton field. Maps of the quality variation in a cotton field can then be used with yield-variation and cost maps to create a detailed map of the profit variation within a cotton field. Thus, the sensor was designed with future adaptation to a harvester in mind. Thus, the design uses components that can be powered by the 12-VDC system on a harvester. The distance from the camera lens to the sample window is 300 mm (11.8 in.). The camera is mounted with its centerline coincident to the vertical centerline of the sensor's frame (figure 17), and the light sources are mounted 355 mm (14 in.) from the sample window to prevent illumination problems. The field of view of the camera on the cotton sample is 64 mm x 79.5 mm. The frame has a 305-mm (12-in.) square footprint and is 457 mm (18 in.) tall, not including the camera, to allow sufficient area within the frame to place the light source outside of the reflected field of view of the camera.

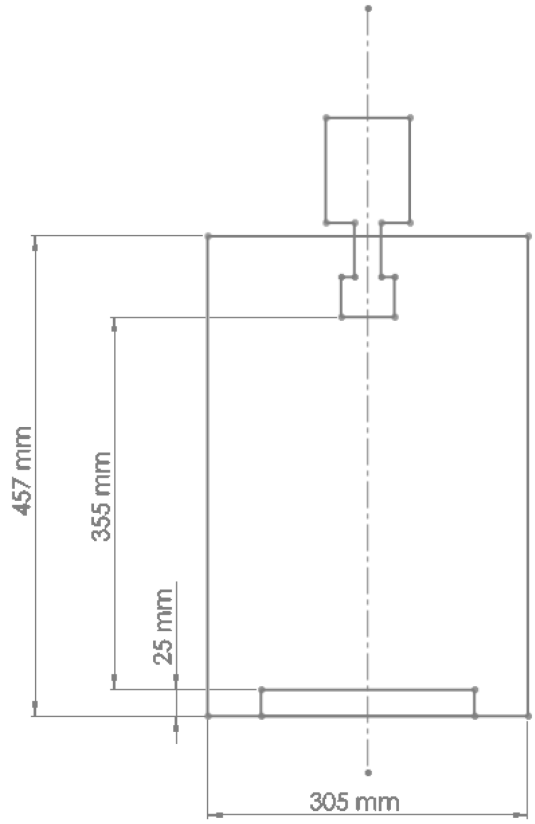


Figure 17: Initial sketch of prototype frame dimensions displaying components alignment.

Prototype

A prototype was built for laboratory testing. A steel frame (figure 18) was incorporated to allow this prototype to be adapted to a harvester. A motorized filter wheel was attached to the camera (figure 19) to allow automation of the filter changing process, and the optics were assembled in such a way that each pixel represents 0.0625 mm^2 of the cotton sample.

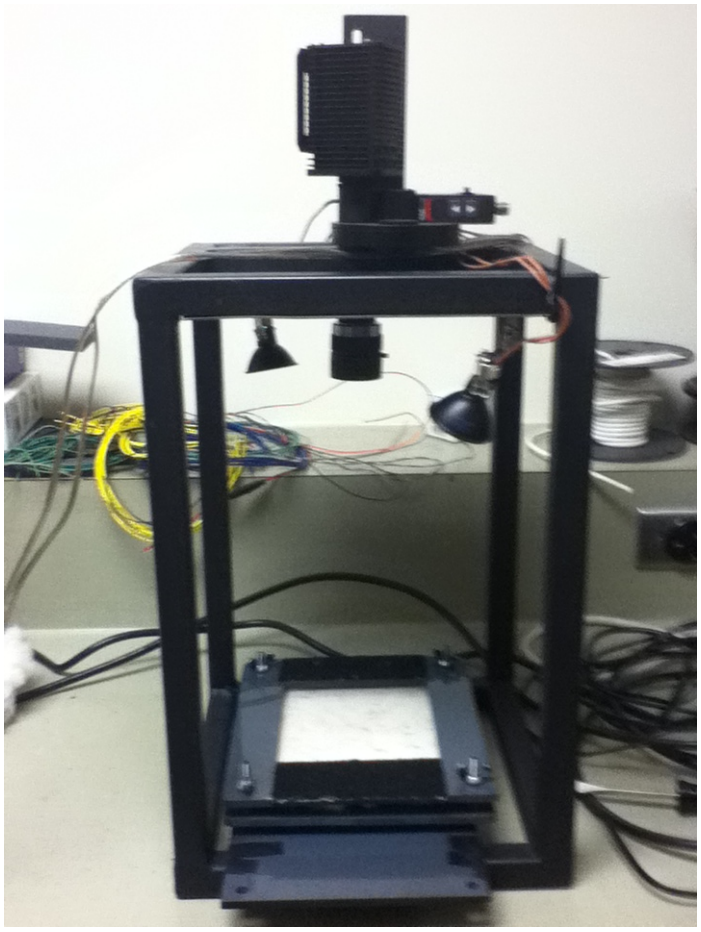


Figure 18: Completed prototype frame with sampler window, camera assembly, and light source.

Figure 19: Alpha NIR camera with motorized filter wheel attached.

Calculations indicated that the 650-nm and 1450-nm filters provided for roughly equal camera sensitivity, which also happened to be the lowest level out of the 5 filter-detector combinations. The 1300-nm filter required an ND filter with only 21% transmission, and the 1550-nm and 1600-nm filers required ND filters with 79% and 70% transmission, respectively (Table 1). These filter modifications roughly balanced the camera sensitivity across all the filters, so that apertures and integration times could be held fairly constant. Even though no significant distortion due to refraction should occur due to different thicknesses of the optical filters, a clear glass "blank" was attached to each to equalize the thickness of the filter combinations.

Table 1: Neutral density requirements on commercially available optical band-pass filters

Filter Part Number	%T
FB650-40	100%
FB1300-30	21%
FB1450-12	100%
FB1550-12	79%
FB1600-12	70%

The sensor control program gives the user visual feedback as the program runs in the laboratory. The user must initialize the program once and then enter a sample ID for each sample as it is presented to the system. After the sample ID is entered, the control program uses the camera and the motorized filter wheel to take images in the appropriate wavebands before prompting the user that it is finished with the current sample. While the user removes the current sample and prepares the next, the control program stores the images that the camera has just acquired onto non-volatile memory and runs a processing algorithm to identify trash and cotton fiber in the sample, and then it measures the reflectance of the cotton fiber in the appropriate NIR wavebands.

Samples were processed in the laboratory at a rate of one every 45 seconds. This processing rate was limited by the time it took to remove one sample from the sample holder and prepare the next. Overall the operation of the prototype worked just as expected. No problems were encountered during the data collection process.

Relationship of Prototype Measurements to HVI Measurements

The image processing method of using fixed threshold limits on the ratio of the 650 and 1300 nm images to classify pixels as "cotton" or "other" during sampling showed a large amount of error while reviewing the system data. The average reflectivity values calculated with this method showed no significant correlation to the HVI micronaire data ($R^2 = 0.1$). The sample summaries created during sampling indicated that some samples used less than two percent of the image data to calculate the averages. This was caused by a lack of flexibility in the boundaries of the classification threshold.

On the other hand, when classifying pixels with the dynamic threshold limit method and mode procedure on ratio images, the reflectance of the fiber portion of the seed cotton (the effects of the trash in the image having been removed) had a strong relationship (R^2=0.73) with the measured micronaire value (table 2). The relationship between lint cotton reflectance and HVI micronaire was even stronger (R^2=0.86; table 2). It is clear that the predicted values for seed cotton had a linear relationship with the observed values (figure 20). The residuals in this model were observed to be normally distributed (figures 21 and 22). With this method of trash removal from seed cotton, half of the measurements were within 0.2 units of the standard HVI micronaire measurement. This method used an average of 58% of the pixels in each image (Figure 23) to calculate the reflectance of cotton fiber in the NIR wavebands of interest.

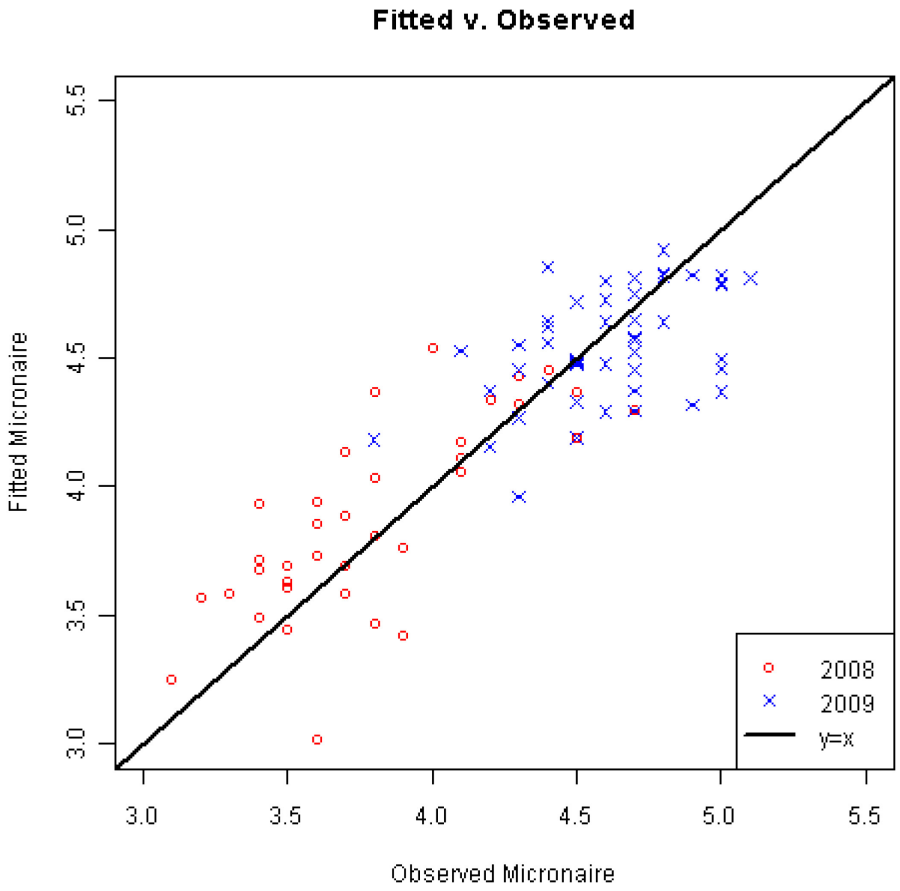


Figure 20: Predicted v. observed – ratio image method; seed cotton

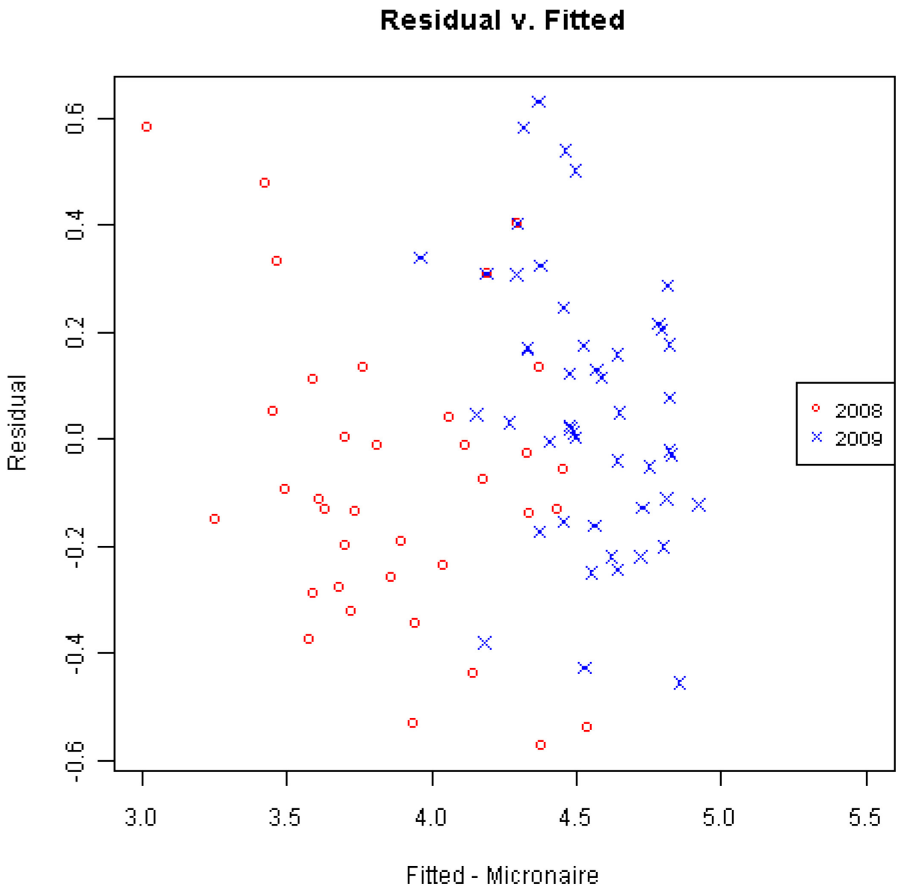


Figure 21: Residual v. fitted – ratio image method; seed cotton

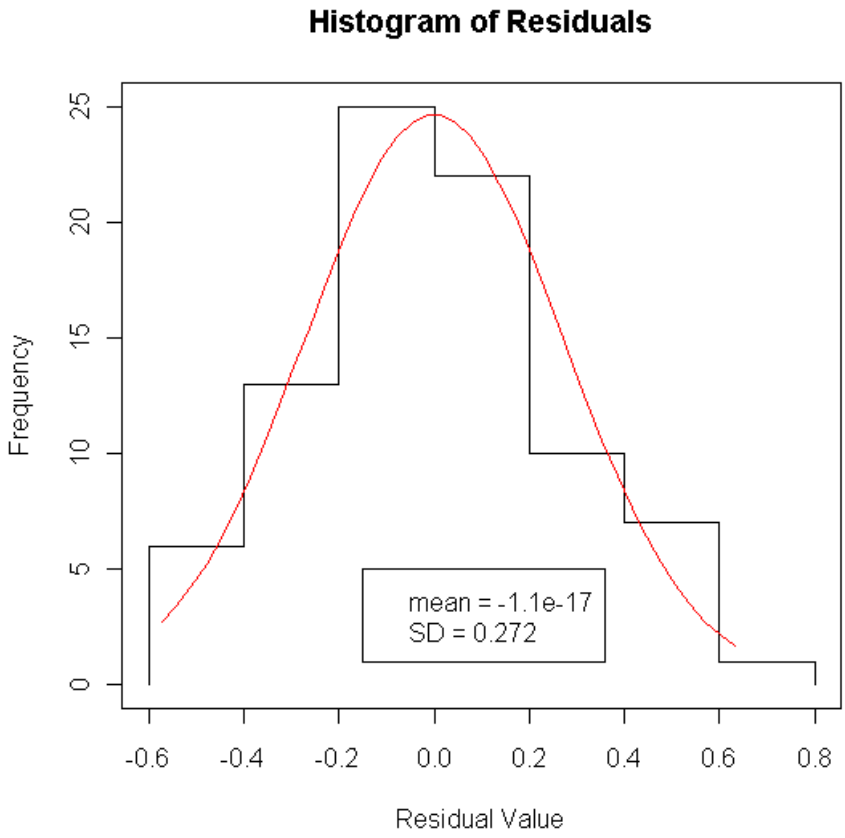


Figure 22: Distribution of residuals – ratio image method; seed cotton

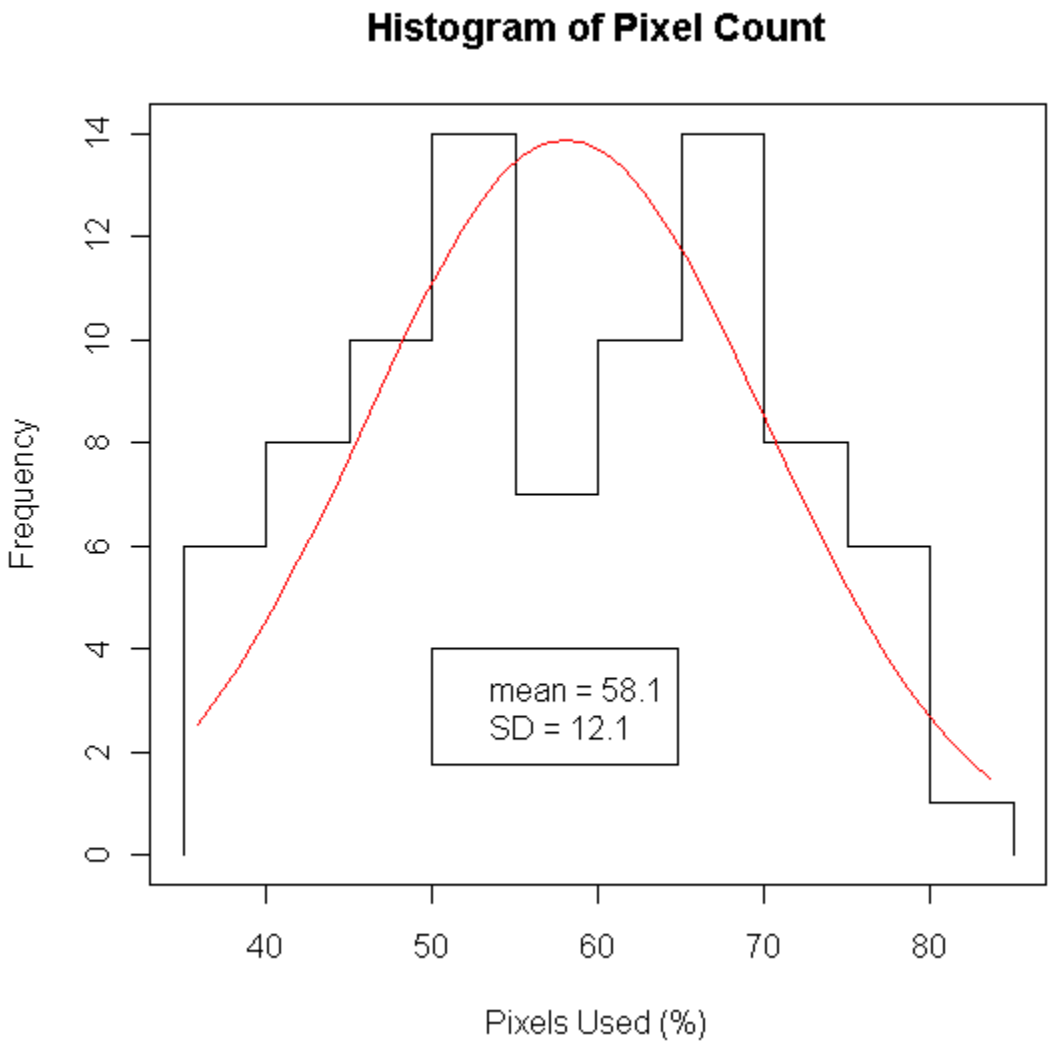


Figure 23: Pixel count distribution – ratio image method; seed cotton

When classifying pixels with the dynamic threshold limit method and mode procedure on a single-image (650nm), a slightly stronger correlation (R^2=0.74) between seed cotton NIR reflectance and HVI micronaire was observed (table 2). This method was also applied with the cotton samples after they had been ginned, and ratio images had a slightly stronger relationship (R^2=0.86) than single images (R^2=0.85; table 2). The predicted values from the seed cotton model were plotted against the observed values

(figure 24), and the residuals of this model were also found to be normally distributed (figures 25 and 26). This method used an average of 81% of the pixels in each image (Figure 27) to calculate the reflectance of cotton fiber in the NIR wavebands of interest.

Table 2: Linear regression results. Generated using both seed-cotton and lint-cotton, and both the single-image and image-ratio method of trash removal.

		Single Image Method 650nm image			Image Ratio Method		
		R^2	Adj. R^2	RMSE	R^2	Adj. R^2	RMSE
Seed Cotton	1450 + 1550 + 1600	0.74	0.73	0.27	0.73	0.72	0.28
	1450 + 1550	0.73	0.72	0.27	0.72	0.71	0.28
	1550 + 1600	0.73	0.73	0.27	0.72	0.71	0.28
	1450 + 1600	0.74	0.73	0.27	0.72	0.71	0.28
	1450	0.68	0.67	0.30	0.66	0.65	0.31
	1550	0.73	0.73	0.27	0.71	0.71	0.28
	1600	0.73	0.73	0.27	0.71	0.71	0.28
Lint Cotton	1450 + 1550 + 1600	0.85	0.85	0.20	0.86	0.86	0.20
	1450 + 1550	0.85	0.85	0.20	0.86	0.86	0.20
	1550 + 1600	0.85	0.85	0.20	0.86	0.86	0.19
	1450 + 1600	0.84	0.84	0.21	0.85	0.85	0.20
	1450	0.80	0.80	0.23	0.81	0.81	0.23
	1550	0.85	0.85	0.20	0.86	0.86	0.20
	1600	0.84	0.84	0.21	0.85	0.85	0.20

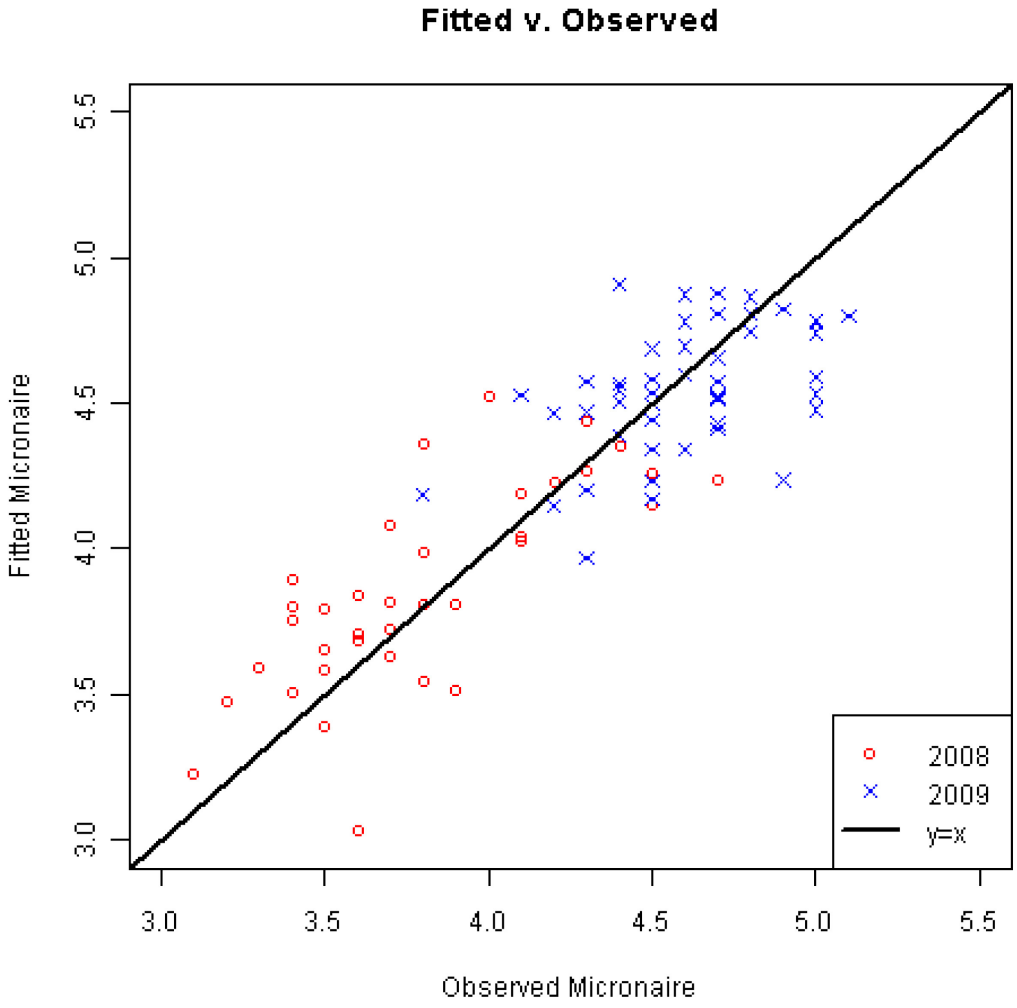


Figure 24: Fitted v. observed - single image method; seed cotton

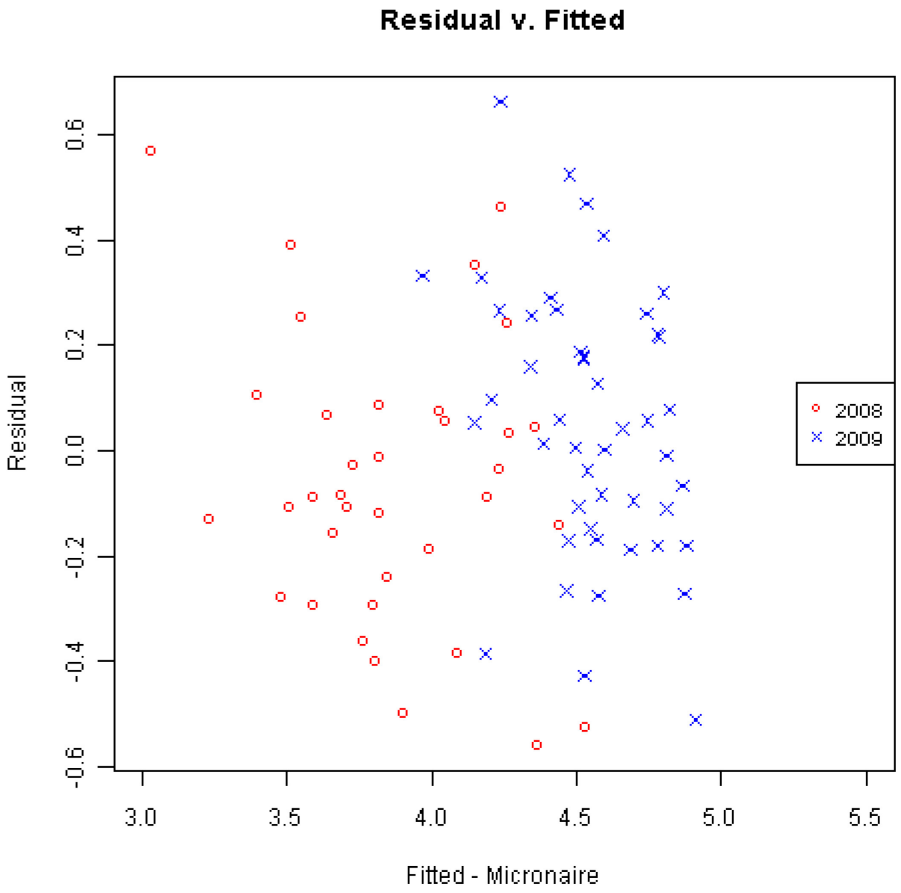


Figure 25: Residual v. fitted – single image method; seed cotton

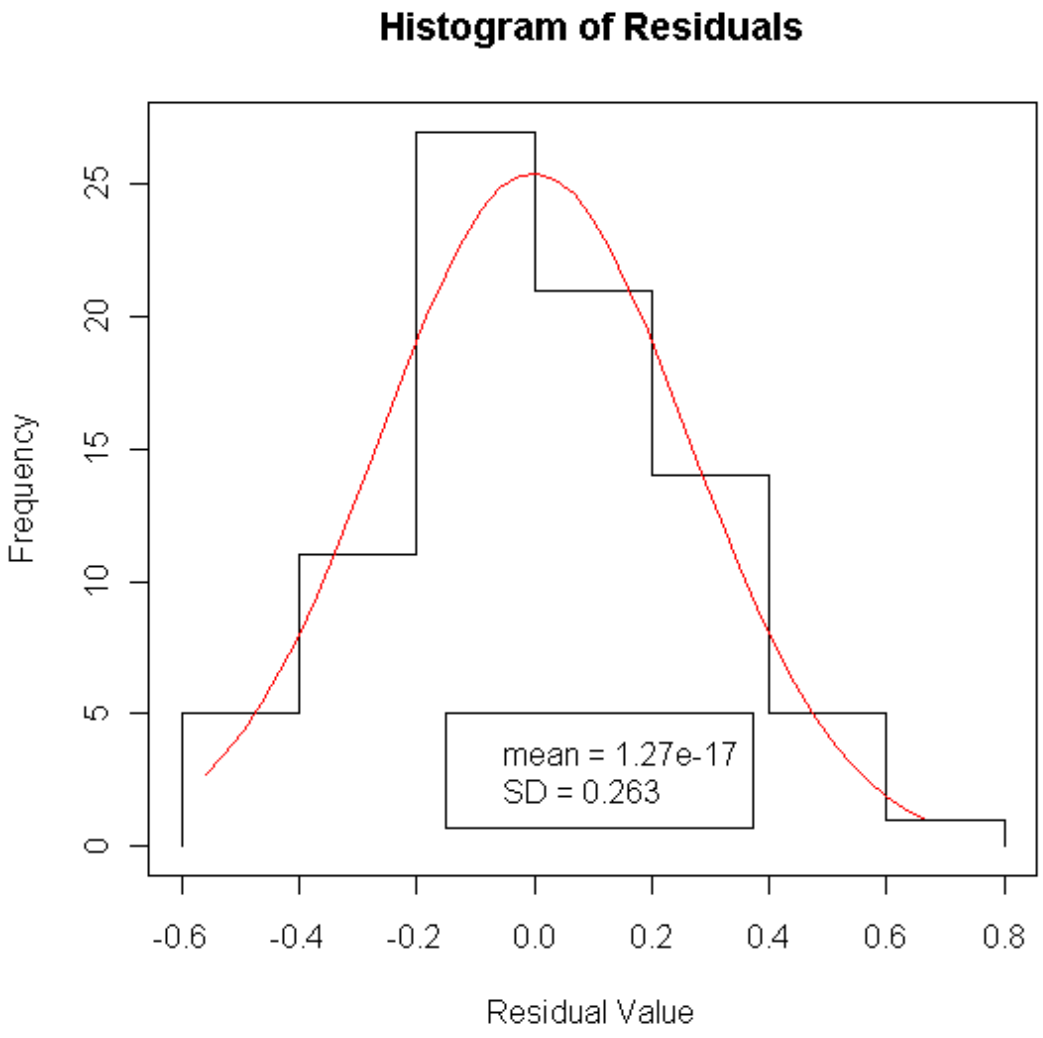


Figure 26: Distribution of residuals – single image method; seed cotton

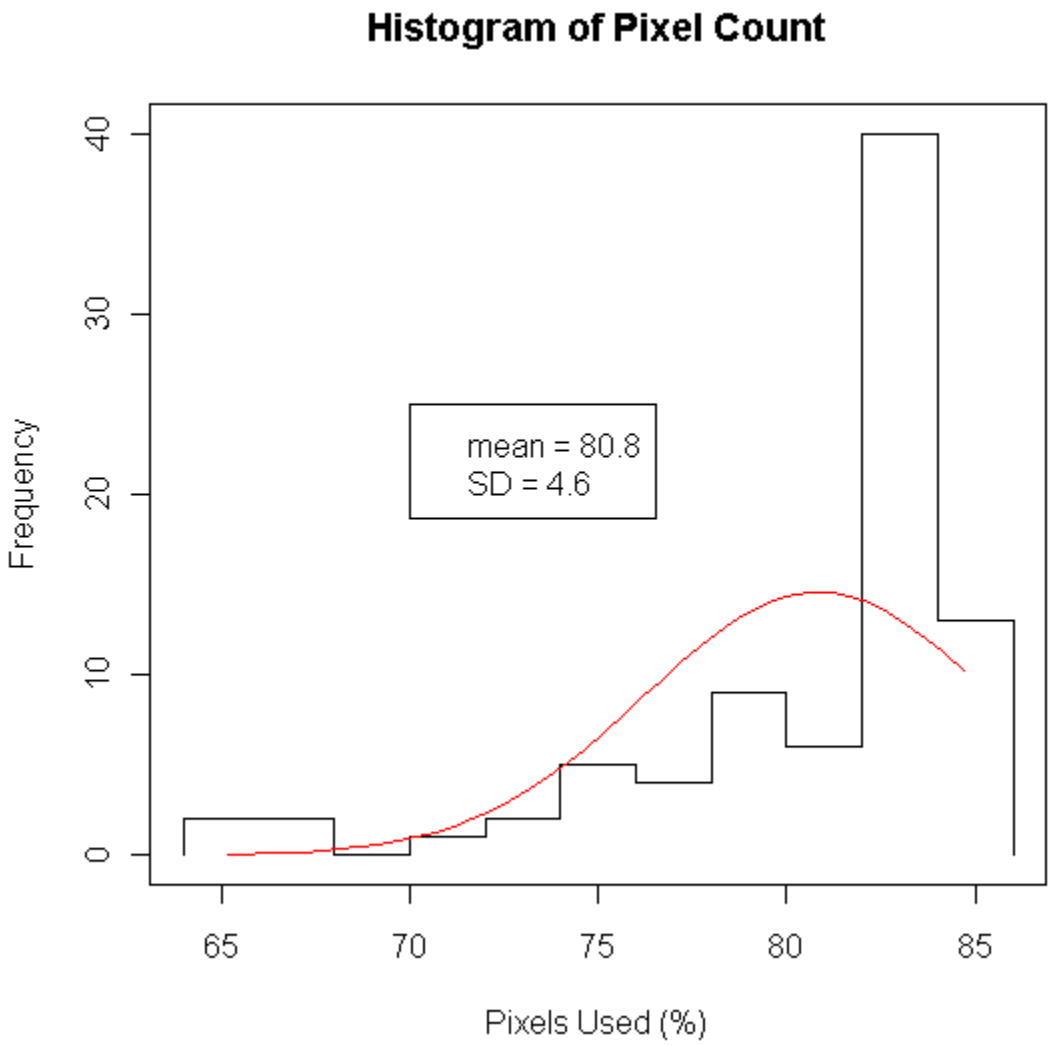


Figure 27: Distribution of pixel count – single image method; seed cotton

For the ratio-image approach, the best threshold limit identification parameters excluded 10% of the pixels from the lower value side of the histogram, and then excluded pixels more than 1.5 times farther away from the mode on the higher side. All of the pixels that fall between the two boundaries are classified as cotton lint.

The optimum settings for the single-image method were found to be when the upper boundary is set to exclude at least 12% of the pixels from the high value end of the histogram, and then use a 1:1 ratio of distance from upper limit to mode to set the lower limit.

Methods of object detection were considered, but attempts at manually implementing edge detection or a local threshold were unsuccessful due to the lack of contrast in the images collected with the prototype sensor (figure 28). This contrast issue was due to low camera sensitivity in the visible spectrum, and the similarity of NIR reflectance between leaves, sticks, and cotton fiber.

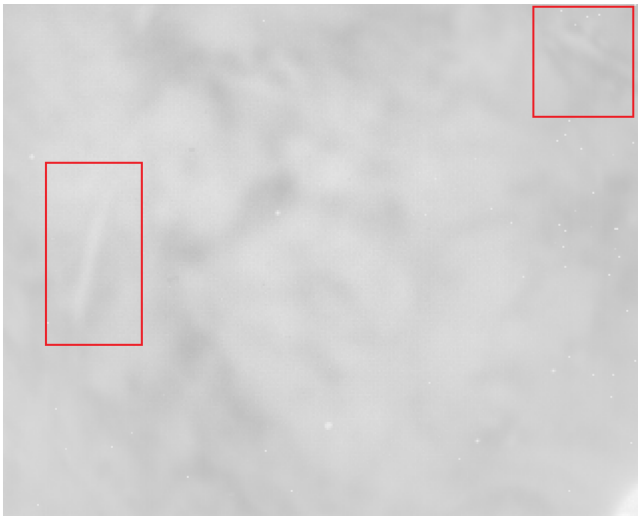


Figure 28: 650-nm seed cotton image. Note the low contrast between cotton fiber and trash. In each box, there is a stick within the cotton fiber. There are also ground pieces of leaf scattered throughout the sample which are not detectable in the image.

System Improvement Possibilities

The level of prediction accuracy of this image-based sensing system has room to be improved, first by improving the radiometric resolution of the image sensor. While the camera used ostensibly has 12-bit resolution, it was discovered during testing that the

camera could not make use of the full 4,096 values for each pixel. When exposed to a saturating light source, the pixels can all be saturated and provide a value near 4,095, but when the sensor is exposed to no light at all, the sensor still registers pixel values near 2,000 instead of the expected values close to zero. This problem diminished the radiometric resolution of the camera by roughly half and to an extent hindered system accuracy. A second improvement that might improve accuracy is a better image-collection system. While a camera with a VisGaAs detector was chosen for its sensitivity to both visible and near-infrared light, the Alpha NIR camera was not nearly as sensitive in the visible spectrum as in the NIR spectrum. This great difference in sensitivity has an effect on the radiometric resolution of the images by limiting the filters that can be used to longer wavelengths. Since one sensor is taking images in both portions of the spectrum, the energy available to the sensor in the visible range may not be adequate to provide a high dynamic range for measurement in visible wavebands. The sensor's only useful sensitivity in the visible spectrum is in the "red" area. While it is easy to look at a seed cotton sample and identify small leaf particles and find the edges of sticks and large leaves, images taken by the Alpha NIR camera indicate these objects are not as easy to identify with an image that lacks blue and green sensitivity. By using a separate "visible" sensor to detect and identify trash in a seed cotton sample, the trash could be more identifiable in the images. Separating the visible and NIR sensors, however, would introduce the issue of pixel alignment, as the pixels in the visible image must correspond with pixels in the NIR images in order to mask out the NIR pixels that do not represent cotton fiber. By using a visible image, other methods of trash detection that were unsuccessfully implemented with this prototype, such as object detection, could be reevaluated.

Practical Implications

A comparison of calibration methods by Gibson (2001) using both calibration cottons and orifices showed that an HVI system is capable of reproducing results within 0.1 micronaire units 82% of the time using orifice calibration, and 78% using cotton

calibration. It is worth noting that the environment for HVI analysis is very controlled, requiring 21±1°C with 65±2% relative humidity (ASTM 2008). These restrictions would be difficult to follow with an in-situ micronaire measurement system, and were not intentionally followed during laboratory testing. If this system is adapted to operate in the field during a harvest and take real-time in-situ measurements of cotton fiber quality, it can be used to create high-resolution fiber-quality maps of a cotton field. Data from this system could be used in conjunction with a wireless module tracking system (Ge, 2007; Sjolander, 2009) to track quality variability within module-harvest areas, as well as to validate the measurements of the fiber-quality measurement system. If the micronaire value at each point within each module is combined with the yield rate, the weighted average micronaire within the module boundaries should equal the HVI measured micronaire for the module. Using this type of data along with a yield map and a cost map, a profit map could be created and site-specific crop-management methods could be improved, ultimately increasing the profitability and environmental stewardship of a cotton farming operation. By taking measurements of cotton fiber micronaire value during harvest, cotton can even be separated based on micronaire value in the harvesting operation, creating modules with increased micronaire uniformity that could garner a higher premium for the farmer.

CONCLUSIONS

A system was designed that can acquire and process images of machine-harvested seed cotton and relate the reflectance values in the images to the measured micronaire value of the cotton fiber. Currently cotton samples must be manually presented to the prototype system, but this process can be automated, allowing the system to repeatedly collect seed cotton samples during harvest, and to acquire and process images of them to determine micronaire.

The prototype system operated as designed, and it shows great potential for being adapted to field testing in the future. During bench testing, 50% of the predicted micronaire values were within 0.2 micronaire units of the HVI measured value, while 90% of the predicted values were within 0.4 units of the measured values. The prototype requires the user to calibrate the system once per session, and then simply present a cotton sample to the image sensor and enter a sample name on the computer. The computer then acquires images of the sample presented to it, processes the images by removing the effects of the trash particles, then averages the reflectance before asking the user for the next sample.

This system shows promise in taking in-situ measurements of cotton fiber quality during harvest. With adequate adaptations to make this prototype suitable for field work, and the addition of an automatic sampling device, this system will be able to create detailed fiber quality spatial variability maps without the impractical time and labor requirements of manual methods.

REFERENCES

ASTM Standard D1448, 2011, "Standard Test Method for Micronaire Reading of Cotton Fibers," ASTM International, West Conshohocken, PA, 2011, DOI: 10.1520/D1448-11, www.astm.org

ASTM Standard D1776, 2008, "Standard Practice for Conditioning and Testing Textiles," ASTM International, West Conshohocken, PA, 2008, DOI: 10.1520/D1776-08E01

Calhoun, D. S., T. P. Wallace, W. S. Anthony, and M. E. Barfield. 1996. Comparison of lint fraction and fiber quality data from hand – vs machine – harvested samples in cotton yield trials. In *Proc. Beltwide Cotton Conf.*, Memphis, Tenn.: Natl. Cotton Council of Am., 611-615.

Chakraborty, K., and D. Ethridge. 1999. Cotton quality price differentials from textile mills' perspective: an update. In *Proc. of the Beltwide Cotton Conf.*, Memphis, Tenn.: Natl. Cotton Council of Am., 256-259.

Cotton Inc. 2010. The classification of cotton. Cary, N. C.: Cotton Incorporated. Available at: www.cottoninc.com/classificationofcotton. Accessed 31 December 2010.

Faulkner, W. B., J. D. Wanjura, B. W. Shaw, and E. F. Hequet. 2008. Effect of harvesting methods on fiber and yarn quality from irrigated cotton on the high plains. ASABE Paper No. 083283. St. Joseph, Mich.: ASABE.

Ge, Y. 2007. Mapping in-field cotton fiber quality and relating it to soil moisture. PhD diss. College Station, Tex.: Texas A&M University.

Ge, Y., J. A. Thomasson, R, Sui, C. L. S. Morgan, S. W. Searcy, and C. B. Parnell. 2008. Spatial variation of fiber quality and associated loan rate in a dryland cotton field. *Precision Agric.* 9(4):181-194.

Ge, Y., S. Stanislav, and C. Morgan. 2009. Relationships between cotton lint yield, fiber quality, and soil apparent electrical conductivity. In *Proc. of the Beltwide Cotton Conf.*, Memphis, Tenn.: Natl. Cotton Council of Am., 1272-1276.

Gibson, L. 2001. Improving micronaire accuracy and precision. In *Proc. of the 14th Annual EFS System Conf.,* Greenville, S. C.: Cotton Inc., 83-91.

Johnson, R. M., R. G. Downer, J. M. Bradow, P. J. Bauer, and E. J. Sadler. 2002. Variability in cotton fiber yield, fiber quality, and soil properties in a southeastern coastal plain. *Agronomy Journal* 94(6): 1305-1316.

Sassenrath, G.F., E.R. Adams, and J.R. Williford. 2005. Rapid sampling system for determination of cotton fiber quality spatial variability. *Applied Eng. Agric.* 21(1): 9-14.

Sassenrath, G. F., F. Suminto, D. to, and J. Ray Williford. 2006. Automated cotton sampler for determination of fiber quality spatial variability. ASABE Paper No. 061178. St. Joseph, Mich., ASABE.

Sjolander, A. J. 2009. Automation of a wireless cotton module tracking system for cotton fiber quality mapping. M.S. thesis. College Station, Tex.: Texas A&M University.

Stanislav, S. M. 2010. A field-scale assessment of soil-specific seeding rates to optimize yield factors and water use in cotton. M.S. thesis. College Station, Tex.: Texas A&M University.

Stanislav, S. and C. Morgan. 2007. Poster: Evaluating electrical conductivity based management zones for cotton lint quality. Madison Wis.: ASA Annual International Meeting.

Sui, R., J. A. Thomasson, Y. Ge, and C. Morgan. 2008. Multispectral sensor for in-situ cotton fiber quality measurement. ASABE Paper No. 083735. St. Joseph, Mich., ASABE.

Thomasson, J.A. and S.A. Shearer. 1995. Correlation of NIR data with cotton quality characteristics. *Trans. of the ASAE* 38(4): 1005-1010.

Thomasson, J.A. and R. Sui. 2000. Unpublished data provided by the authors from their work conducted at Mississippi State University.

Thomasson, J. A., and R. Sui. 2004. Optical-Reflectance-Based Mass-Flow Sensor. Patent number 6,809,821. Washington, D.C.: USPTO.

Walker, J. 2004. VisGaAs camera sees red… and more. R&D Magazine March 2004. Available at: www.corebyindigo.com. Accessed Jan. 5th, 2009.

APPENDIX A

IMAGE DATA

Appendix A-1: Data for Lint Cotton using the Image Ratio Method (10%, 1.5:1)

(mean and standard deviation of pixel reflectance provided for each waveband of interest, along with the measured micronaire value and the number of pixels classified as cotton fiber for the sample)

		1450 nm		1550 nm		1600 nm			
ID	year	mean	SD	mean	SD	mean	SD	mic	#px
I00	2008	2844.789	17.6	2783.926	18.0	2783.069	16.3	4.5	23864
I01	2008	2858.685	19.7	2803.345	17.6	2804.272	16.6	4.3	31451
I02	2008	2860.111	24.0	2818.760	21.6	2821.527	20.6	3.4	29097
I03	2008	2839.710	20.7	2783.475	19.8	2779.016	18.9	4.1	33955
I04	2008	2841.565	15.9	2778.938	16.5	2787.977	15.9	4.4	25858
I05	2008	2886.810	17.7	2830.014	18.9	2825.298	16.9	3.6	34346
I06	2008	2837.531	18.9	2804.257	18.7	2806.420	17.8	3.8	13275
I07	2008	2891.314	17.5	2840.882	17.6	2839.299	15.4	3.1	34086
I08	2008	2883.380	18.0	2818.497	18.5	2822.614	17.0	3.5	31560
I09	2008	2878.238	14.7	2816.118	15.2	2812.591	14.0	3.7	34802
I10	2008	2875.097	12.9	2809.534	14.0	2819.159	12.7	3.6	25373
I11	2008	2873.720	20.9	2811.280	19.7	2817.291	18.3	3.4	31384
I12	2008	2906.511	13.4	2847.471	13.6	2835.476	12.5	3.3	29564
I13	2008	2895.784	16.5	2841.649	16.1	2840.601	15.0	3.4	32371
I14	2008	2887.930	17.8	2824.825	16.3	2821.403	16.8	3.4	24147
I15	2008	2901.528	15.5	2842.495	15.2	2836.762	14.0	3.5	19998
I16	2008	2885.983	26.5	2825.748	26.5	2823.909	24.3	3.5	39631
I17	2008	2897.793	17.8	2837.098	17.3	2844.402	19.7	3.2	33312
I18	2008	2890.218	23.8	2829.950	24.2	2826.947	22.1	3.5	37242
I19	2008	2857.360	27.6	2792.138	25.1	2792.225	24.5	3.8	24113
I20	2008	2867.612	18.2	2805.350	17.3	2808.937	16.3	3.7	34164
I21	2008	2855.447	16.5	2791.306	16.5	2793.696	14.8	4.5	26039

I22	2008	2849.583	21.0	2786.251	22.5	2792.677	26.9	4.7	42016
I23	2008	2851.595	17.3	2794.787	16.7	2792.895	15.3	4.2	29322
I24	2008	2871.608	20.3	2803.693	18.0	2813.033	19.3	4.1	31772
I25	2008	2870.201	17.9	2802.363	15.8	2805.399	15.4	3.9	21810
I26	2008	2867.002	18.4	2809.351	17.1	2813.638	16.3	3.6	34325
I27	2008	2871.635	22.2	2809.168	20.8	2813.751	20.3	3.8	38357
I28	2008	2845.712	16.7	2789.643	15.6	2792.476	14.9	3.9	32295
I29	2008	2879.202	14.6	2809.758	14.9	2811.479	13.9	3.7	34879
I30	2008	2887.490	17.6	2815.982	18.6	2810.274	17.6	4.0	34352
I31	2008	2850.925	18.1	2795.157	18.3	2783.066	16.9	4.1	30531
I32	2008	2867.568	21.5	2805.799	19.4	2806.460	19.0	3.7	34668
I33	2008	2845.240	17.7	2780.969	19.2	2786.091	17.0	4.3	34878
I34	2008	2872.041	23.4	2808.703	24.3	2805.803	21.6	3.8	37230
I35	2008	2869.293	44.2	2804.332	38.3	2812.390	36.3	3.6	30911
101	2009	2810.890	20.0	2771.722	17.9	2765.469	16.3	4.4	33424
102	2009	2822.029	19.8	2774.250	18.8	2770.405	17.1	4.5	29934
103	2009	2847.119	17.0	2797.600	16.2	2790.706	15.7	4.3	22674
104	2009	2827.968	19.8	2773.996	18.2	2781.283	17.2	4.3	27851
111	2009	2812.355	19.9	2757.873	19.0	2767.537	17.4	4.5	38940
112	2009	2844.469	22.2	2793.491	21.6	2799.785	20.0	4.3	31025
113	2009	2833.872	23.1	2788.034	21.3	2791.453	20.7	4.2	24281
114	2009	2809.949	17.6	2764.309	15.1	2763.452	14.8	4.4	30560
121	2009	2838.495	17.8	2784.366	15.9	2782.451	15.0	4.5	32803
122	2009	2815.407	16.4	2767.258	16.4	2772.079	15.1	4.5	33430
123	2009	2836.992	22.1	2780.230	19.4	2784.344	18.7	4.3	36217
124	2009	2818.435	18.9	2762.382	17.6	2766.978	16.9	4.6	34752
131	2009	2822.390	19.5	2765.882	16.5	2773.458	16.5	4.7	25912
132	2009	2804.056	19.7	2754.075	20.2	2758.144	18.1	4.4	33868

133	2009	2839.082	18.9	2791.514	17.0	2794.143	16.1	4.2	31350
134	2009	2840.596	22.2	2796.735	20.3	2788.908	19.4	3.8	32754
201	2009	2817.197	23.1	2763.673	23.4	2766.555	20.8	4.7	34312
202	2009	2807.965	18.1	2760.636	16.3	2766.072	15.5	4.5	21855
203	2009	2816.270	16.2	2756.983	15.5	2759.337	14.6	4.7	19492
204	2009	2803.130	25.3	2745.590	21.5	2749.533	20.9	5.0	31540
211	2009	2811.329	20.6	2758.233	20.7	2761.413	18.5	4.8	17445
212	2009	2800.477	20.7	2749.765	20.3	2741.842	19.3	4.7	40571
213	2009	2816.438	15.5	2774.412	15.7	2770.395	14.4	4.6	28885
214	2009	2792.636	22.0	2753.901	20.1	2752.003	18.4	4.8	27891
221	2009	2796.010	19.3	2739.181	19.7	2737.929	17.5	5.0	29749
222	2009	2801.403	20.8	2750.905	19.6	2748.103	18.5	4.7	32226
223	2009	2792.531	18.3	2729.063	17.1	2740.773	15.9	5.0	31564
224	2009	2809.372	17.3	2759.288	16.6	2754.262	15.8	4.6	28796
231	2009	2814.896	18.3	2758.343	17.5	2769.369	15.9	5.0	22845
232	2009	2818.199	24.4	2754.140	21.6	2768.005	20.2	4.8	22758
233	2009	2811.225	15.2	2759.040	15.0	2753.882	13.8	4.9	32032
234	2009	2783.508	19.6	2735.035	18.8	2735.873	17.1	5.0	24961
301	2009	2828.927	18.1	2783.422	18.1	2776.771	16.2	4.1	14405
302	2009	2817.922	15.9	2759.373	15.8	2771.087	14.6	4.5	16728
303	2009	2806.036	18.6	2748.128	18.1	2749.556	16.9	4.6	32204
304	2009	2820.707	21.4	2770.067	18.0	2774.101	17.0	4.5	22454
311	2009	2793.080	19.1	2734.901	18.2	2740.247	16.4	4.7	22903
312	2009	2818.075	21.1	2766.391	19.6	2766.382	19.0	4.8	30421
313	2009	2811.908	23.3	2751.883	23.0	2751.924	21.6	4.7	32954
314	2009	2831.857	17.0	2767.143	15.8	2765.392	15.0	4.7	18069
321	2009	2795.196	22.3	2742.067	21.0	2749.993	19.3	5.1	32891
322	2009	2836.775	19.9	2786.609	18.9	2779.707	18.1	4.6	39231

323	2009	2812.580	19.7	2753.415	19.8	2758.905	18.4	4.5	39535
324	2009	2803.285	24.9	2739.523	22.1	2745.396	21.0	5.0	25182
331	2009	2831.845	21.2	2778.450	21.2	2781.030	19.3	4.4	34966
332	2009	2821.997	21.6	2763.042	20.1	2763.596	19.3	4.9	30716
333	2009	2838.334	15.5	2783.561	14.9	2790.624	14.2	4.4	32446
334	2009	2809.933	26.1	2754.174	23.4	2760.475	22.4	4.7	34674

Appendix A-2: Data for Seed Cotton using the Image Ratio Method (10%, 1.5:1)

(mean and standard deviation of pixel reflectance provided for each waveband of interest, along with the measured micronaire value and the number of pixels classified as cotton fiber for the sample)

		1450 nm		1550 nm		1600 nm			
ID	year	mean	SD	mean	SD	mean	SD	mic	#px
I00	2008	2824.642	52.2	2771.427	46.4	2774.194	44.7	4.5	45013
I01	2008	2799.896	63.4	2757.259	53.0	2760.079	51.9	4.3	56561
I02	2008	2835.798	64.4	2784.373	59.0	2793.104	56.2	3.4	60592
I03	2008	2826.658	67.7	2774.696	56.8	2779.687	53.1	4.1	58881
I04	2008	2797.592	63.3	2754.615	53.4	2749.355	50.9	4.4	54345
I05	2008	2853.074	56.2	2805.854	53.2	2803.621	49.6	3.6	39492
I06	2008	2744.896	117.2	2741.448	95.0	2739.836	89.7	3.8	64915
I07	2008	2884.974	60.8	2848.926	50.8	2831.392	47.6	3.1	60138
I08	2008	2870.665	51.9	2822.370	45.3	2809.751	42.9	3.5	42011
I09	2008	2842.198	49.4	2789.087	45.6	2796.583	42.7	3.7	41486
I10	2008	2911.965	46.1	2859.072	43.6	2858.015	40.3	3.6	45365
I11	2008	2852.836	65.4	2806.394	59.7	2804.467	55.9	3.4	52941
I12	2008	2866.229	54.1	2818.649	49.6	2813.957	45.9	3.3	34574
I13	2008	2870.376	54.5	2824.703	49.8	2820.098	45.3	3.4	64586
I14	2008	2867.166	60.1	2810.503	56.3	2812.319	52.0	3.4	47053
I15	2008	2883.214	50.2	2838.270	46.6	2820.833	42.8	3.5	38829
I16	2008	2868.725	50.3	2817.187	47.3	2814.205	43.0	3.5	42559
I17	2008	2849.479	50.7	2815.244	45.1	2809.359	42.8	3.2	35498
I18	2008	2850.720	75.9	2806.200	68.7	2805.566	64.2	3.5	57745
I19	2008	2866.157	69.2	2818.152	57.0	2824.914	54.1	3.8	41860
I20	2008	2810.351	48.1	2771.774	45.2	2771.248	43.1	3.7	39054
I21	2008	2811.008	52.4	2757.316	47.6	2761.908	44.9	4.5	30369

I22	2008	2809.989	76.4	2759.745	60.0	2765.816	56.4	4.7	53501
I23	2008	2815.114	46.8	2762.760	43.1	2762.675	39.6	4.2	35143
I24	2008	2814.293	56.9	2770.128	49.5	2771.336	46.4	4.1	51468
I25	2008	2855.515	54.2	2804.824	51.8	2802.993	47.7	3.9	55438
I26	2008	2838.032	61.9	2782.628	55.9	2794.437	53.0	3.6	49183
I27	2008	2852.625	51.0	2801.165	46.7	2799.993	43.5	3.8	62733
I28	2008	2877.459	54.7	2832.742	51.7	2824.003	47.6	3.9	41804
I29	2008	2866.817	54.3	2817.252	47.0	2815.299	45.1	3.7	43102
I30	2008	2806.888	60.4	2746.499	56.1	2752.535	53.2	4.0	54318
I31	2008	2828.797	55.1	2774.991	50.0	2785.305	47.6	4.1	51657
I32	2008	2858.077	67.1	2813.632	63.8	2803.902	59.0	3.7	52777
I33	2008	2819.849	53.4	2756.736	49.6	2760.991	47.0	4.3	68061
I34	2008	2839.259	62.3	2782.610	60.0	2786.850	56.1	3.8	58771
I35	2008	2848.350	63.9	2797.202	56.1	2796.520	52.8	3.6	49424
101	2009	2778.540	58.2	2736.415	49.7	2743.528	46.7	4.4	40516
102	2009	2788.601	55.9	2750.528	47.4	2758.690	46.0	4.5	43511
103	2009	2794.900	47.5	2745.697	40.3	2754.462	38.5	4.3	30681
104	2009	2798.823	61.5	2755.684	53.1	2765.459	50.0	4.3	59671
111	2009	2782.979	72.9	2746.926	60.8	2744.613	58.3	4.5	36482
112	2009	2815.807	73.5	2782.281	58.1	2782.030	56.5	4.3	41420
113	2009	2801.760	62.3	2766.690	50.4	2769.186	48.6	4.2	54078
114	2009	2746.066	61.3	2711.309	50.9	2719.440	48.9	4.4	38773
121	2009	2786.752	69.0	2746.830	62.2	2747.317	58.6	4.5	62940
122	2009	2791.368	58.9	2763.481	52.9	2763.236	50.2	4.5	55425
123	2009	2752.753	73.8	2731.896	59.6	2734.888	55.4	4.3	62513
124	2009	2761.314	56.3	2723.825	48.4	2722.937	47.0	4.6	45691
131	2009	2763.145	60.5	2732.983	51.6	2736.303	48.7	4.7	40501
132	2009	2767.425	72.2	2730.206	64.5	2735.219	60.7	4.4	55025

133	2009	2775.276	74.8	2748.083	64.1	2750.031	61.6	4.2	35600
134	2009	2772.596	65.5	2755.671	52.3	2760.191	49.5	3.8	40376
201	2009	2802.264	55.8	2758.583	48.6	2762.680	46.0	4.7	46882
202	2009	2792.058	53.7	2744.744	46.8	2751.897	45.4	4.5	29249
203	2009	2777.913	64.3	2745.622	58.8	2746.119	55.1	4.7	63582
204	2009	2756.922	65.3	2725.267	53.6	2720.280	50.8	5.0	59069
211	2009	2768.563	56.1	2732.023	50.4	2734.571	48.8	4.8	43388
212	2009	2769.472	65.7	2723.771	59.6	2726.044	56.6	4.7	34371
213	2009	2781.177	65.9	2745.433	57.1	2745.748	53.8	4.6	29164
214	2009	2751.751	68.8	2713.643	53.5	2723.615	51.4	4.8	33588
221	2009	2793.081	42.0	2740.803	39.5	2753.201	36.8	5.0	29479
222	2009	2768.039	67.5	2724.153	54.4	2730.361	52.3	4.7	51025
223	2009	2774.398	60.8	2750.516	49.1	2748.331	48.0	5.0	46871
224	2009	2763.167	48.2	2726.939	42.9	2728.029	40.5	4.6	43347
231	2009	2742.751	61.2	2718.459	52.8	2715.576	49.5	5.0	43001
232	2009	2770.840	51.8	2725.603	46.1	2723.872	44.3	4.8	49619
233	2009	2808.037	54.1	2760.048	49.2	2762.795	45.7	4.9	43230
234	2009	2764.848	59.3	2721.980	49.9	2727.472	48.2	5.0	43491
301	2009	2794.395	50.4	2741.555	45.3	2750.741	42.5	4.1	55227
302	2009	2779.668	61.2	2740.320	51.8	2746.918	50.1	4.5	51276
303	2009	2772.500	54.3	2732.527	45.5	2736.372	43.3	4.6	35153
304	2009	2772.815	75.2	2728.677	61.6	2732.263	59.3	4.5	30688
311	2009	2787.791	52.9	2744.240	43.4	2745.730	39.9	4.7	44326
312	2009	2733.603	79.5	2707.611	62.2	2710.030	59.4	4.8	54257
313	2009	2791.673	72.8	2748.846	62.9	2757.489	59.2	4.7	55176
314	2009	2782.940	54.0	2735.676	47.1	2745.468	45.3	4.7	57398
321	2009	2750.849	63.6	2715.433	53.8	2722.561	53.3	5.1	49190
322	2009	2804.485	55.2	2758.209	48.7	2764.496	46.9	4.6	38300

323	2009	2805.372	60.0	2758.725	55.5	2761.326	52.6	4.5	53932
324	2009	2777.723	55.5	2743.649	48.9	2746.834	45.8	5.0	47133
331	2009	2783.254	50.3	2736.438	43.6	2740.703	41.3	4.4	40261
332	2009	2766.719	58.8	2724.126	53.6	2723.531	49.6	4.9	51222
333	2009	2802.184	49.0	2752.621	45.1	2757.367	42.4	4.4	36870
334	2009	2782.868	49.6	2737.666	42.2	2736.980	41.2	4.7	55506

Appendix A-3: Data for Lint Cotton using the Single Image Method (16%, 2.4:1, 650nm)

(mean and standard deviation of pixel reflectance provided for each waveband of interest, along with the measured micronaire value and the number of pixels classified as cotton fiber for the sample)

		1450 nm		1550 nm		1600 nm			
ID	year	mean	SD	mean	SD	mean	SD	mic	#px
I00	2008	2843.545	22.2	2781.562	22.3	2781.687	20.6	4.5	67824
I01	2008	2852.672	21.9	2796.650	22.1	2798.447	20.6	4.3	68530
I02	2008	2857.353	25.3	2814.654	23.6	2817.898	22.7	3.4	68268
I03	2008	2831.900	23.6	2774.623	24.0	2771.738	22.5	4.1	68075
I04	2008	2839.840	16.5	2777.430	17.2	2786.582	16.1	4.4	68123
I05	2008	2880.686	19.5	2822.889	20.9	2819.311	19.1	3.6	68129
I06	2008	2835.243	27.9	2803.683	24.3	2805.785	24.0	3.8	68486
I07	2008	2883.588	19.9	2832.252	20.2	2831.725	18.2	3.1	68638
I08	2008	2878.724	19.9	2814.462	19.3	2818.600	18.6	3.5	67621
I09	2008	2878.249	17.6	2815.220	17.6	2811.821	16.8	3.7	67723
I10	2008	2872.196	16.4	2806.524	17.3	2816.452	16.2	3.6	68451
I11	2008	2866.882	21.5	2804.290	21.6	2810.712	20.4	3.4	68309
I12	2008	2905.103	22.1	2844.670	19.0	2834.016	18.5	3.3	68582
I13	2008	2896.339	19.3	2841.010	18.0	2840.546	17.2	3.4	69544
I14	2008	2880.996	24.7	2819.481	23.1	2817.210	24.9	3.4	67890
I15	2008	2898.147	18.9	2838.699	19.8	2834.051	18.5	3.5	68932
I16	2008	2875.777	28.7	2815.309	28.1	2814.595	26.0	3.5	68511
I17	2008	2898.002	17.6	2835.734	19.3	2843.175	21.3	3.2	69090
I18	2008	2879.877	30.3	2819.213	30.3	2817.647	27.9	3.5	68536
I19	2008	2853.686	34.1	2788.284	35.4	2789.498	33.6	3.8	69116
I20	2008	2864.729	20.4	2801.333	21.0	2805.758	19.9	3.7	67741
I21	2008	2847.984	20.5	2783.163	20.7	2787.146	19.6	4.5	68495

I22	2008	2843.592	19.4	2779.672	25.6	2787.418	31.7	4.7	68040
I23	2008	2847.407	20.2	2789.929	19.8	2789.179	18.5	4.2	68592
I24	2008	2868.646	18.6	2800.419	19.8	2810.411	21.8	4.1	67541
I25	2008	2873.310	19.9	2804.014	21.1	2808.349	21.1	3.9	67830
I26	2008	2859.257	21.2	2800.056	22.1	2805.475	20.5	3.6	68792
I27	2008	2866.331	24.1	2803.450	23.7	2808.655	23.3	3.8	68385
I28	2008	2843.584	17.3	2786.414	17.6	2789.607	16.6	3.9	67737
I29	2008	2876.241	17.8	2805.908	17.3	2808.357	16.3	3.7	68520
I30	2008	2886.462	21.2	2813.517	21.4	2808.562	20.2	4.0	69023
I31	2008	2848.576	19.1	2791.945	21.7	2781.110	20.4	4.1	68808
I32	2008	2865.870	21.6	2803.690	20.6	2805.173	20.1	3.7	68504
I33	2008	2838.941	22.4	2774.167	21.7	2780.106	19.7	4.3	68036
I34	2008	2865.710	25.0	2801.057	26.5	2799.553	24.0	3.8	68667
I35	2008	2866.652	30.8	2800.798	28.3	2809.854	26.1	3.6	63772
101	2009	2807.213	18.6	2766.075	18.4	2761.179	16.8	4.4	68889
102	2009	2811.946	26.4	2763.424	24.9	2761.110	23.3	4.5	68670
103	2009	2838.945	23.7	2789.331	23.1	2783.867	21.8	4.3	67850
104	2009	2822.845	27.5	2768.771	21.5	2776.185	21.5	4.3	68489
111	2009	2807.379	19.8	2752.244	19.8	2762.421	18.4	4.5	68625
112	2009	2833.382	28.2	2781.784	28.0	2789.609	25.9	4.3	68489
113	2009	2828.680	28.1	2782.595	28.6	2787.304	27.5	4.2	68208
114	2009	2804.019	18.1	2757.552	16.9	2757.941	16.0	4.4	68702
121	2009	2836.132	18.6	2781.299	18.2	2779.921	17.2	4.5	68502
122	2009	2810.984	23.7	2762.261	19.7	2767.792	19.3	4.5	68036
123	2009	2830.114	24.7	2773.469	21.9	2778.477	20.9	4.3	68100
124	2009	2812.743	19.8	2755.971	19.8	2761.257	18.8	4.6	68284
131	2009	2815.601	22.5	2759.100	22.3	2767.997	22.1	4.7	68800
132	2009	2798.203	30.0	2748.355	27.4	2752.976	26.1	4.4	68840

133	2009	2832.129	21.0	2785.151	19.1	2787.877	18.5	4.2	68654
134	2009	2829.249	26.1	2785.518	24.9	2779.164	23.1	3.8	67958
201	2009	2808.271	25.6	2755.528	24.6	2759.119	22.7	4.7	68290
202	2009	2806.142	18.5	2758.439	19.2	2764.770	18.4	4.5	67855
203	2009	2810.204	20.0	2750.572	19.7	2754.103	18.4	4.7	69010
204	2009	2799.039	24.9	2741.039	22.7	2745.815	21.7	5.0	69003
211	2009	2809.082	30.4	2756.430	28.1	2760.392	26.8	4.8	68538
212	2009	2790.907	26.3	2740.074	25.5	2733.250	23.7	4.7	68229
213	2009	2807.926	19.8	2766.114	19.4	2763.197	18.1	4.6	68716
214	2009	2784.430	25.8	2745.329	25.3	2745.075	23.1	4.8	68339
221	2009	2786.699	28.1	2729.371	26.5	2729.708	25.4	5.0	68671
222	2009	2794.179	23.5	2743.494	24.6	2741.755	23.0	4.7	67999
223	2009	2782.162	25.4	2719.015	23.5	2731.759	22.2	5.0	68394
224	2009	2802.973	22.9	2752.473	20.5	2748.659	19.3	4.6	67886
231	2009	2808.123	26.5	2752.830	26.5	2764.583	24.4	5.0	68641
232	2009	2809.732	24.9	2746.286	24.0	2760.724	22.8	4.8	68187
233	2009	2809.220	22.2	2756.774	18.6	2751.978	18.0	4.9	68194
234	2009	2777.064	21.7	2728.462	22.0	2730.285	20.3	5.0	68623
301	2009	2821.642	32.4	2776.513	29.4	2771.500	28.4	4.1	68802
302	2009	2813.450	17.7	2754.238	18.3	2766.781	17.3	4.5	68578
303	2009	2797.474	22.7	2739.822	21.2	2742.476	19.6	4.6	68086
304	2009	2813.970	22.4	2763.759	21.5	2768.792	20.2	4.5	68401
311	2009	2784.621	21.8	2727.029	21.9	2733.343	20.1	4.7	67988
312	2009	2815.237	23.0	2762.540	23.1	2763.389	22.4	4.8	68205
313	2009	2806.489	22.8	2745.744	22.3	2746.648	21.2	4.7	68356
314	2009	2825.164	22.4	2761.708	23.1	2760.932	21.3	4.7	68811
321	2009	2786.525	25.1	2733.235	23.2	2742.161	21.7	5.1	67850
322	2009	2831.048	22.8	2779.605	22.4	2773.946	21.0	4.6	68865

323	2009	2805.584	20.2	2745.717	20.9	2752.272	19.5	4.5	68475
324	2009	2794.274	28.6	2731.356	24.9	2738.455	23.9	5.0	68722
331	2009	2825.568	22.7	2771.175	23.4	2775.089	21.5	4.4	68452
332	2009	2817.475	21.5	2757.141	21.4	2758.894	20.1	4.9	68291
333	2009	2831.692	19.5	2775.968	20.2	2784.137	18.8	4.4	68083
334	2009	2802.212	26.7	2746.380	24.7	2753.559	23.7	4.7	68078

Appendix A-4: Data for Seed Cotton using the Single Image Method (16%, 2.4:1, 650nm)

(mean and standard deviation of pixel reflectance provided for each waveband of interest, along with the measured micronaire value and the number of pixels classified as cotton fiber for the sample)

		1450 nm		1550 nm		1600 nm			
ID	year	mean	SD	mean	SD	mean	SD	mic	#px
I00	2008	2805.491	61.9	2756.852	56.0	2759.161	53.6	4.5	68294
I01	2008	2791.866	58.5	2752.213	54.1	2754.802	51.8	4.3	63601
I02	2008	2827.184	57.4	2777.012	54.2	2785.906	51.0	3.4	65725
I03	2008	2821.008	62.5	2770.672	51.2	2775.395	48.0	4.1	64880
I04	2008	2799.756	55.3	2754.716	48.3	2750.319	45.1	4.4	54895
I05	2008	2847.272	62.0	2800.933	56.2	2798.581	53.8	3.6	68098
I06	2008	2739.640	75.5	2734.078	67.6	2733.141	62.7	3.8	54395
I07	2008	2875.405	60.8	2841.678	51.0	2824.239	48.6	3.1	67279
I08	2008	2865.629	52.5	2817.621	46.8	2805.548	44.8	3.5	68512
I09	2008	2834.916	47.7	2784.574	43.9	2791.502	41.3	3.7	66640
I10	2008	2899.399	53.1	2848.113	47.6	2847.084	45.7	3.6	68311
I11	2008	2837.354	64.0	2793.576	57.1	2792.166	54.2	3.4	68175
I12	2008	2851.659	56.2	2807.294	49.3	2804.246	45.4	3.3	63966
I13	2008	2857.915	51.2	2814.174	48.1	2809.825	44.4	3.4	65023
I14	2008	2844.682	65.1	2791.218	56.9	2793.798	54.2	3.4	67796
I15	2008	2876.430	52.1	2833.642	48.0	2816.658	44.2	3.5	64205
I16	2008	2857.202	49.5	2804.496	48.2	2803.374	43.9	3.5	67238
I17	2008	2847.093	57.4	2812.893	50.3	2807.829	48.9	3.2	68937
I18	2008	2831.582	71.1	2789.224	62.9	2789.322	58.7	3.5	68253
I19	2008	2840.560	80.1	2799.209	63.1	2806.845	60.0	3.8	68138
I20	2008	2803.590	56.5	2766.929	51.9	2766.208	49.6	3.7	68545
I21	2008	2815.539	68.0	2765.071	62.3	2767.723	59.9	4.5	68342

I22	2008	2803.738	73.9	2755.357	58.0	2760.587	56.0	4.7	66415
I23	2008	2812.358	59.5	2762.179	53.4	2761.438	50.8	4.2	67651
I24	2008	2800.281	61.9	2759.784	51.3	2760.528	49.6	4.1	68438
I25	2008	2840.160	60.4	2791.952	55.2	2790.285	52.0	3.9	67486
I26	2008	2847.531	49.5	2790.417	47.8	2802.480	44.0	3.6	53125
I27	2008	2838.449	56.2	2791.722	50.1	2789.551	47.8	3.8	67466
I28	2008	2858.300	62.0	2816.237	56.8	2808.619	54.2	3.9	68738
I29	2008	2847.269	61.9	2803.241	53.5	2801.181	50.7	3.7	67871
I30	2008	2796.585	63.5	2738.822	57.4	2744.433	55.4	4.0	67345
I31	2008	2818.135	56.5	2767.081	51.4	2777.224	48.5	4.1	67688
I32	2008	2846.136	70.4	2803.121	65.1	2793.742	61.2	3.7	68378
I33	2008	2808.009	55.5	2747.868	49.7	2751.354	47.5	4.3	67326
I34	2008	2832.118	55.5	2777.021	53.3	2781.300	49.4	3.8	61182
I35	2008	2840.208	57.7	2789.754	51.8	2789.352	48.7	3.6	67187
101	2009	2767.488	58.4	2728.033	50.4	2734.954	47.7	4.4	68549
102	2009	2787.192	48.9	2747.953	43.9	2757.223	41.0	4.5	58678
103	2009	2769.493	57.8	2725.648	48.6	2735.125	46.2	4.3	67245
104	2009	2794.091	54.3	2751.001	48.2	2761.067	44.7	4.3	61753
111	2009	2772.009	61.9	2737.386	54.3	2736.070	51.1	4.5	67295
112	2009	2803.985	66.0	2771.939	53.8	2772.397	51.0	4.3	66727
113	2009	2791.916	56.8	2757.254	48.4	2761.087	45.0	4.2	62337
114	2009	2729.436	58.9	2697.247	51.6	2706.219	48.7	4.4	66018
121	2009	2790.651	55.9	2748.855	53.2	2750.318	48.7	4.5	53051
122	2009	2780.089	59.0	2754.807	52.9	2754.977	49.5	4.5	67339
123	2009	2750.999	57.2	2729.413	49.9	2732.333	46.4	4.3	63932
124	2009	2741.624	61.0	2708.576	51.5	2708.743	49.4	4.6	67940
131	2009	2752.752	56.9	2723.880	50.5	2728.341	46.9	4.7	60630
132	2009	2767.879	59.7	2731.120	53.6	2736.584	49.3	4.4	60015

133	2009	2752.189	71.4	2729.891	59.2	2733.122	56.1	4.2	67960
134	2009	2764.716	57.6	2746.385	49.9	2751.266	46.6	3.8	62293
201	2009	2777.840	64.0	2739.676	51.7	2744.015	50.0	4.7	68017
202	2009	2762.704	66.3	2725.024	55.9	2731.587	53.7	4.5	67903
203	2009	2760.802	64.9	2731.774	59.1	2732.560	55.1	4.7	68745
204	2009	2752.270	56.4	2719.677	50.2	2714.776	47.5	5.0	64728
211	2009	2747.913	61.4	2714.738	54.2	2718.127	51.1	4.8	63042
212	2009	2750.181	61.6	2708.964	55.5	2711.986	52.0	4.7	66864
213	2009	2749.588	70.2	2719.223	62.0	2720.216	58.7	4.6	65223
214	2009	2737.968	58.5	2701.193	48.9	2711.395	45.7	4.8	61307
221	2009	2772.799	50.5	2723.238	46.4	2736.594	43.6	5.0	67789
222	2009	2746.357	71.0	2709.214	54.1	2715.320	51.5	4.7	66913
223	2009	2754.799	60.9	2733.030	49.1	2731.752	47.6	5.0	67596
224	2009	2746.729	58.3	2713.985	48.4	2715.180	47.1	4.6	67840
231	2009	2731.228	62.4	2711.745	51.5	2708.892	49.2	5.0	68412
232	2009	2762.276	53.1	2719.410	45.2	2717.315	44.1	4.8	68427
233	2009	2802.504	57.5	2756.150	52.7	2759.668	49.1	4.9	68792
234	2009	2756.818	57.8	2716.288	49.4	2721.593	46.8	5.0	67458
301	2009	2781.396	51.0	2731.565	45.0	2741.070	42.3	4.1	68452
302	2009	2760.322	63.1	2726.389	51.5	2733.282	49.1	4.5	68656
303	2009	2763.767	56.9	2725.776	48.5	2730.666	45.9	4.6	68126
304	2009	2761.173	62.7	2720.335	52.8	2725.599	49.4	4.5	65069
311	2009	2772.247	60.1	2734.482	44.2	2735.407	41.7	4.7	67674
312	2009	2739.074	54.7	2709.584	46.6	2711.754	44.1	4.8	57141
313	2009	2775.765	66.0	2734.966	58.5	2743.874	55.2	4.7	63184
314	2009	2776.529	51.1	2729.614	44.7	2740.242	42.5	4.7	61052
321	2009	2737.728	61.5	2706.009	54.4	2713.768	52.1	5.1	64648
322	2009	2786.341	59.0	2743.809	50.0	2750.773	48.1	4.6	68312

323	2009	2783.447	60.3	2740.744	55.1	2743.700	51.3	4.5	68081
324	2009	2758.346	62.3	2728.043	52.4	2731.833	49.5	5.0	68144
331	2009	2774.616	51.8	2730.569	43.4	2735.271	41.2	4.4	68557
332	2009	2755.211	57.2	2715.685	51.2	2715.106	47.4	4.9	67831
333	2009	2789.896	51.8	2743.882	45.8	2749.107	42.6	4.4	67783
334	2009	2770.877	54.3	2728.709	44.9	2727.832	43.9	4.7	68201

APPENDIX B-1

C++ CODE

```
1   //   Program main function
2
3   #include "f_specs.h"       // header file containing all function declarations
4   #include <time.h>
5   #include <iomanip>
6
7   using namespace std;
8
9   #define SIZE 26
10
11  int main(int argc, char* argv[]) {
12
13    time_t seconds = time(NULL);        // reads the system time when the program starts
14    char startTime[SIZE];               // storage for date & time string
15    ctime_s(startTime, SIZE, &seconds); // get date & time string
16
17    HANDLE hPort = 0;                   // required for serial communication
18    DWORD dwNumBytesWritten;
19
20    do{
21      string port = getPort();          // gets the port name from user
22      hPort = openPort(port);           // opens the serial port
23    } while(!hPort);
24
25    INTERFACE_ID iid;                   // NI-IMAQ interface ID
26    SESSION_ID sid;                     // NI-IMAQ session ID
27    imgInterfaceOpen("img0", &iid);     // Opens NI-IMAQ interface with desired device
28    imgSessionOpen(iid, &sid);          // Opens NI-IMAQ session with desired interface
29
30    WriteFile(hPort, &"sensors=0\r", 10, &dwNumBytesWritten, NULL);    // turns filter wheel position sensors off while idle
31    cameraSetup(sid);                   // Initializes the camera
```

```
32
33      char drive = getDrive();        // Gets drive letter from user
34
35      const vector<dblImage> offset = loadAdjustMap(drive);     // Loads pixel offset table created by the Calibration program.
36
37
38      char* dataFilename = getFilename(drive, dirDATA, "RESULTS.TXT");
39      ofstream outfile;
40
41      outfile.open(dataFilename, ios::app);
42      outfile << "----- " << startTime << endl;
43      outfile << "sampID 1450nm 1550nm 1600nm" << endl;
44      outfile << fixed << setprecision(3);
45
46      do {
47        vector<dblImage> images;                // clean set of sample images
48        vector< vector<image> > rawCollection;    // set of raw image collection from camera
49        string sampID = getID();                // gets the sample ID from user
50
51        if (sampID == "QUIT") {                 // Exits program if the user enters QUIT for the sample ID
52          imgClose(iid,1);                    // Closes the IMAQ interface
53          CloseHandle(hPort);                 // Closes serial port
54          outfile << endl;
55          outfile.close();
56          return 0;
57        }
58        string path = getPath(drive, dirIMAGES, sampID);     // generates the storage path for the images collected
59
60        cout << "Taking images: ";
61        for(int i=0; i<NO_FILTERS; ++i) {
62          char* filename = getFilename(path, sampID, FILTERS[i], ".PNG"); // filename for image currently being collected
```

```
63          changeFilter(i,hPort);                          // changes the filter wheel to the desired position
64          setIntegrationTime(sid, INT_TIMES[i]);          // sets the integration time of the camera
65          Sleep(500);                                     // pauses to allow the filter wheel to move
66          vector<image> rawSet = snapSet(iid, sid, filename);    // collects a raw set of images through a single filter
67          rawCollection.push_back(rawSet);                // stores the raw sets of images for a single sample
68          cout << (i+1) << " - ";
69        }
70        changeFilter(0,hPort);        // puts the first filter back into position
71        cout << "DONE" << endl;
72
73        for(uint8_t i=0; i<rawCollection.size(); ++i) {
74          dblImage I = average(rawCollection.at(i));          // cleans the raw images by averaging images that should be identical
75          applyOffset(I,offset.at(i));                        // applies the offset
76          uint16_t* buffer = I.as_int();
77          imgSessionSaveBufferEx(sid, buffer, I.get_filename());  // saves buffer as image onto hard storage using given filename
78          saveBufferBin(I);                                   // saves buffer as binary data
79          saveImgDec(I);                                      // saves buffer as numerical data
80          images.push_back(I);                                // stores images of the sample for processing.
81        }
82
83        vector<double> SValues = processImages(images, sampID);
84
85        outfile << sampID << " " << SValues.at(0) << " " << SValues.at(1) << " " << SValues.at(2) << endl;
86
87      } while (true);   // continuous loop until QUIT is entered for the sample ID
88    }
```

```
1   // applyOffset.cpp
2
3   /*
4       This function applies an offset map (O) to an image (I)
5       The pixel values in I are divided by the corresponding pixel value in O
6
7       both I and O are objects of the custom class dblImage
8   */
9
10  #include "f_specs.h"
11
12  using namespace std;
13
14  void applyOffset(dblImage& I, const dblImage& O){
15
16    for (int i=0; i<I.size(); ++i)
17      I.data[i] = (I.data[i] / O.data[i]);
18  }
```

```
1   // average.cpp
2
3   /*
4     This function averages a vector of images passed into the function call.
5
6     Each element of the vector should an image of an identcal object.
7     The purpose of this function is to reduce noise at the image sensor.
8   */
9
10  #include "f_specs.h"
11
12  using namespace std;
13
14  dblImage average(const vector<image>& imageSet) {
15
16    dblImage I;                                 // user defined class
17    double n = imageSet.at(0).size();                 // image size
18    int m = imageSet.size();                      // number of images to average
19
20    for(int i=0; i<n; ++i){                       // pixel loop
21      double sum= 0;                              // initalize sum to zero
22      for(int k=0; k<m; ++k)                        // image loop
23        sum += imageSet.at(k).data[i];                // sum the values of each corresponding pixel
24      I.data[i] = sum/m;                          // divide the sum by the number of images
25    }                                         // move to the next pixel
26
27    I.set_filename(imageSet.at(0).get_filename());     // transfer the filename to the averaged image
28
29    return I;                                   // return the averaged image
30  }
```

```
1   // binImage.cpp
2
3   /*
4     This is the implementation of the binImage class
5
6     This class is used as a container for a  binary image
7     Each pixel can only contain a value of true or false.
8   */
9
10  #include "binImage.h"
11
12  binImage::binImage(const int& r, const int & c) : _rows(r),    // default constructor
13                                          _cols(c),    // allocates memory for an image
14                                          _count(0),   // containing r*c pixels
15                                          len(0) {     // default values: r=256 c=318
16    filename = new char[len+1];
17    filename[len] = 0;
18
19    data = new bool[_rows * _cols];
20    for(int i=0; i<(_rows * _cols); ++i) {
21      data[i] = false;
22    }
23  }
24
25  binImage::binImage(const binImage& o) : _rows(o._rows),        // copy constructor
26                                  _cols(o._cols),        // copies an existing image of type binImage
27                                  _count(o._count),      // into new memory
28                                  len(o.len) {
29    filename = new char[len+1];
30    for(int i=0; i<len; ++i)
31      filename[i] = o.filename[i];
```

```
32      filename[len] = 0;
33
34      data = new bool[_rows * _cols];
35      for(int i=0; i<o.size(); ++i)
36        data[i] = o.data[i];
37    }
38
39    binImage::~binImage() {          // destructor
40      delete [] filename;            // clears memory longer in use
41      delete [] data;
42    }
43
44    int binImage::rows() const {     // returns the number of rows of pixles in the image
45      return _rows;
46    }
47
48    int binImage::cols() const {     // returns the number of columns of pixles in the image
49      return _cols;
50    }
51
52    int binImage::size() const {     // returns the total number of pixels in the image
53      return (_rows * _cols);
54    }
55
56    char* binImage::get_filename() const {     // returns the filename associated with the image
57      return filename;
58    }
59
60    void binImage::set_filename(const char* str) {     // sets the filename associated with the image
61      delete [] filename;                 // to the value (str) passed into the function call
62      len = strlen(str);                  // after deleting the old filename and unallocating
```

```
63      filename = new char[len+1];                    // assocated memory
64      for(int i=0; i<len; ++i)
65        filename[i] = str[i];
66      filename[len] = 0;
67    }
68
69    int binImage::count() const {              // returns the number of pixels with a value of true
70      return _count;
71    }
72
73    void binImage::add(const int& x) {         // sets a specified pixel to true
74      if(!data[x]){
75        data[x] = true;
76        _count ++;
77      }
78    }
79
80    void binImage::subtract(const int& x) {    // sets a specified pixel to false
81      if(data[x]){
82        data[x] = false;
83        _count --;
84      }
85    }
86
87    void binImage::reset() {                   // sets all pixels to false
88      for(int i=0; i<(_rows * _cols); ++i)
89        data[i] = false;
90      _count = 0;
91    }
```

```
1   // binImage.h
2
3   /*
4     This is the class declaration file for the class binImage.
5
6     For detailed descriptions of the functions, see binImage.cpp
7   */
8
9   #ifndef BINIMAGE_H
10  #define BINIMAGE_H
11
12  #include <cstdint>
13  #include <string>
14
15  class binImage{
16  public:
17    binImage(const int& r = 256,
18           const int& c = 318);
19    binImage(const binImage&);
20    ~binImage();
21
22    int rows() const;
23    int cols() const;
24    int size() const;
25
26    char* get_filename() const;
27    void set_filename(const char* str);
28
29    int count() const;
30    void add(const int&);
31    void subtract(const int&);
```

```
32      void reset();
33
34      bool* data;
35
36    private:
37      char* filename;    // variable to hold the filename associated with the image
38      int _rows;         // number of rows of pixels in the image
39      int _cols;         // number of columns of pixles in the image
40      uint32_t _count;   // counts the number of true pixels int he image
41      int len;           // variable used to set the length of the filename
42    };
43    #endif
```

```
1  // cameraSetup.cpp
2
3  /*
4    This function sends a pre-set serial command to the Alpha-NIR camera
5    to initalize it for use.
6  */
7
8  #include "f_specs.h"
9
10 using namespace std;
11
12 void cameraSetup(const SESSION_ID& sid) {
13
14   uInt32 size = 10;                                  // size of the char* to be sent through the serial port
15   unsigned char* buffer = new unsigned char[size];          // char* containing the serial command
16   uint16_t sum = 0;                                   // variable containing the checksum
17
18   buffer[0] = 0x49;                          // Byte 1: 0x49 (ASCII character I) must start all serial communication
19   buffer[1] = 0x03;                          // Bytes 2&3: 0x0300 Read Long Integration Mode
20   buffer[2] = 0x00;
21   buffer[3] = 0xFF;                          // Byte 4: Status from slave
22   buffer[4] = 0x00;                          // Byte 5: Packet Count (for debugging)
23   buffer[5] = 0x00;                          // Byte 6&7: number of data bytes
24   buffer[6] = 0x01;
25   buffer[7] = 0x00;                          // Byte 8: data
26   for(uint8_t i=0; i<(size - 2); ++i)
27     sum += buffer[i];                        // sums the bytes in the serial command
28   buffer[size - 2] = sum / 0x100;                  // Byte 9&10: checksum
29   buffer[size - 1] = sum % 0x100;
30
31   imgSessionSerialFlush(sid);                          // Empties the Serial communications buffer
```

```
32      imgSessionSerialWrite(sid,(char*)buffer,&size,3000);       // sends the serial command to the open session
33      imgSessionSerialReadBytes(sid,(char*)buffer,&size,3000);   // reads the response from the slave
34      delete [] buffer;                                  // deletes the buffer from memory
35    }
```

```
1   // changeFilter.cpp
2
3   /*
4     This function changes the position of the FW102B filter wheel through serial communication
5
6     The serial hardware does not provide a response, so no provisions are given to read one.
7   */
8
9   #include "f_specs.h"
10
11  using namespace std;
12
13  void changeFilter(const int& n, const HANDLE& hPort) {
14
15      DWORD dwNumBytesWritten;      // necessary for serial communication
16      string buffer = "pos=";       // string containg the command to send to the filter wheel
17      buffer += (n+49);             // converts the desired position number into a character; adds to string
18      buffer += '\r';               // adds carriage return to string
19
20      WriteFile (hPort, buffer.c_str(), 6, &dwNumBytesWritten, NULL);   // sends serial command
21  }
```

```
1   // createMask.cpp
2
3   /*
4     This function creates a binary Mask image based on threshold limits passed to it.
5
6     This fuction was used in this research only to show that a method of pixel classification
7     can be implemented in the program.  The method of classification shown in this function
8     creates a mask image based on fixed limits.  Methods of dynamic pixel classification were
9     programmed in MATLAB (Appendix B).
10  */
11
12  #include "f_specs.h"
13
14  using namespace std;
15
16  binImage createMask(const dblImage& A, const dblImage& B, const double& low, const double& high){
17
18    binImage mask;                          // allocate memory for a binary image
19
20    dblImage C;                             // allocate memory for a new intensity image
21    for(int i=0; i<C.size(); ++i)
22       C.data[i] = A.data[i] / B.data[i];          // create an intensity image based on a ratio
23                                            // of the two images passed into the function call
24    for(int i=0; i<C.size(); ++i){
25     if ((C.data[i] < high) && (C.data[i] > low))   // classify pixels using the threshold boundaries
26       mask.add(i);                         // passed into the function call and populate the
27    }                                       // binary image
28
29    string filename = A.get_filename();             // pass the filename on to the binary image
30    int len = filename.length();
31    filename.erase(len-8);                  // replace the extension of the filename with
```

```
32      filename += ".MSK";                         // .MSK to give the binary image a unique
33      mask.set_filename(filename.c_str());            // filename
34
35      return mask;                            // return the binary image
36    }
```

```
1   // dblImage.cpp
2
3   /*
4     This is the implementation file for the user defined dblImage class.
5
6     This class contains an r*c array of double values that represent pixels in
7     and intensity image.
8
9     There is also a container for a filename so the image data can be saved externally
10  */
11
12  #include "dblImage.h"
13
14  dblImage::dblImage(const int& r, const int& c) : _rows(r),   // default constructor
15                                                   _cols(c),   // default values: r=256 c=318
16                                                   len(0) {    // filename is empty
17    filename = new char[len+1];
18    filename[len] = 0;
19
20    data = new double[_rows * _cols];
21    for(int i=0; i<(_rows * _cols); ++i)
22      data[i] = 65535;
23  }
24
25  dblImage::dblImage(const dblImage& o) : _rows(o._rows),    // copy constructor
26                                          _cols(o._cols),    // copies the object passed
27                                          len(o.len) {       // into new memory
28    filename = new char[len + 1];
29    for(int i=0; i<len; ++i)
30      filename[i] = o.filename[i];
31    filename[len] = 0;
```

```
32
33      data = new double[_rows * _cols];
34      for(int i=0; i<o.size(); ++i)
35        data[i] = o.data[i];
36    }
37
38    dblImage::~dblImage() {           // destructor
39      delete [] filename;             // releases memory no longer used
40      delete [] data;
41    }
42
43    int dblImage::rows() const{       // returns the number of rows of pixels
44      return _rows;
45    }
46
47    int dblImage::cols() const{       // returns the number of columns of pixels
48      return _cols;
49    }
50
51    int dblImage::size() const{       // returns the number of pixels
52      return (_rows * _cols);
53    }
54
55    char* dblImage::get_filename() const{   // returns the filname associated with the image
56      return filename;
57    }
58
59    void dblImage::set_filename(const char* str) {    // sets the filename associated with the image
60      delete [] filename;                     // to the value passed into the function
61      len = strlen(str);
62      filename = new char[len + 1];
```

```
63      for(int i=0; i<len; ++i)
64        filename[i] = str[i];
65      filename[len] = 0;
66    }
67
68    uint16_t* dblImage::as_int() const {              // returns a 16 bit integer array containing
69      uint16_t* o = new uint16_t[_rows * _cols];      // the pixel values rounded to the nearest integer
70
71      for(int i=0; i<(_rows * _cols); ++i)
72        o[i] = data[i] + .5;
73
74      return o;
75    }
```

```
1   // dblImage.h
2
3   /*
4     This is the class declaration file for the class dblImage
5
6     For details about each class function see the class implementation file
7     (dblImage.cpp)
8
9     This class was designed to hold a single image from an Alpha-NIR camera (FLIR Technologies, Wilsonville, OR)
10
11    a double precision variable was used to allow the image to be filtered by using an average
12    of multiple captures of the same image.
13  */
14
15  #ifndef DBLIMAGE_H
16  #define DBLIMAGE_H
17
18  #include <string>
19  #include <cstdint>
20
21  class dblImage{
22  public:
23    dblImage(const int& r = 256,
24          const int& c = 318);
25    dblImage(const dblImage&);
26    ~dblImage();
27
28    int rows() const;
29    int cols() const;
30    int size() const;
31
```

```
32      char* get_filename() const;
33      void set_filename(const char*);
34
35      uint16_t* as_int() const;
36
37      double* data;    // contains the pixel data for the image
38
39    private:
40      char* filename;  // contains the filename associated with the image
41      int _rows;       // contains the number of rows of pixels
42      int _cols;       // contains the number of columns of pixels
43      int len;         // contains the lenght of the filename
44    };
45    #endif
```

```
1  // f_specs.h
2
3  /*
4    This is the common header file for the program.
5
6    All user defined functions and classes are declared in this file and then implemented in
7    separate .cpp files.
8
9    Constants and definitons are declared in this file to ease modification.
10
11   To see details about any function, please refer to the corresponding implementation file
12 */
13
14 #include "niimaq.h"       // NI-IMAQ header file.  Required for using the PCI-1422 card
15                           // to communicate with the camera.  This file is provided by
16                           // National Instruments (Austin, TX)
17 #include <cstdint>        // required to used uint(n)_t types
18 #include <iostream>
19 #include <string>
20 #include <Windows.h>
21 #include <vector>
22 #include <fstream>
23 #include "image.h"        // required to use the image class
24 #include "dblImage.h"     // required to use the dblImage class
25 #include "binImage.h"     // required to use the binImage class
26
27 #define NO_FILTERS 6      // defines the number of positions on the filter wheel
28 #define REPS 5            // defines the number of captures of the same image to average
29                           // to filter the signal, reducing noise
30 #define ROWS 256          // defines the number of rows of pixels in an image
31 #define COLS 318          // defines the number of columns of pixels in an image
```

```
32
33          //-----  Constant declaration ----------//
34      // constant array containing the names of each filter
35  const std::string FILTERS[NO_FILTERS] = {"_650", "_1300", "_1450", "_1550", "_1600", "_750LP"};
36      // constant array containing the integration time value for each filter
37  const uint16_t INT_TIMES[NO_FILTERS] = {51252, 50510, 49445, 50000, 50000, 51296};
38      // constant string for the directory containing system calibration data
39  const std::string dirCALIBRATION = ":\\DATA\\CALIBRATION\\";
40      // constant string for the folder containing the sample images
41  const std::string dirIMAGES = ":\\DATA\\SAMPLES\\";
42  const std::string dirDATA = ":\\DATA\\";
43
44  char getDrive();                    // requests a drive letter from the user
45
46  std::string getID();                // requests a sample identification number from the user
47
48  std::string getPath(const char&,          // assembles a path passed variables
49                      const std::string&,
50                      const std::string&);
51  std::string getPath(const char&,          // overloading of the previous function
52                      const std::string&);
53
54  char* getFilename(std::string,            // assembles a complete filename from passed variables
55                    const std::string&,
56                    const std::string&,
57                    const std::string&);
58  char* getFilename(std::string,            // overloading of the previous function
59                    const std::string&,
60                    const std::string&);
61  char* getFilename(const char&,            // overloading of the previous function
62                    const std::string&,
```

```
63                  const std::string&);
64
65  dblImage average(const std::vector<image>&);    // averages multiple captures of the same image
66
67  void changeFilter(const int&, const HANDLE&);   // changes the position of the filter wheel
68
69  HANDLE openPort(const std::string&);            // opens the desired COM port with predefined settings
70
71  std::string getPort();                          // requests a port name from the user
72
73  std::vector<image> snapSet(const INTERFACE_ID&, const SESSION_ID&, char*);    // rapidly captures a predefined
74                                                                                 // number of images of the object presented
75
76  void setIntegrationTime(const SESSION_ID&, uint16_t);       // changes the integration time of the image sensor
77
78  void setGain(const SESSION_ID&, const bool&);               // changes the gain of the camera
79
80  void cameraSetup(const SESSION_ID&);                        // initalizes the camera for use
81
82  void saveBufferBin(const dblImage&);                        // saves data in the image buffer as binary data
83
84  void saveImgDec(const dblImage&);                           // saves data in the image buffer as decimal data
85
86  std::vector<dblImage> loadAdjustMap(const char&);           // loads the pixel adjustment map
87
88  void applyOffset(dblImage&, const dblImage&);               // applies offset
89
90  std::vector<double> processImages(const std::vector<dblImage>&,    // run images through the processing algorithm
91                                    const std::string&);
92
93  binImage createMask(const dblImage&,                        // create a binary classification image mask
```

```
94              const dblImage&,
95              const double&,
96              const double&);
```

```
1   // getDrive.cpp
2
3   /*
4     This function requests a drive letter from the user
5   */
6
7   #include "f_specs.h"
8
9   using namespace std;
10
11  char getDrive() {
12
13    char drive = 0;            // initalize character as null
14
15    do{                        // loop while invalid drive letter
16      system("cls");               // clear console
17      cout << "Storage Drive: ";
18      drive = getchar();           // get a single character
19      cin.sync();                  // discard further input
20      cout << endl;
21      if ((drive >= 'a')&&(drive <= 'z'))   // convert lowercase to upper case
22        drive -= 32;
23    }while ((drive < 'A')||(drive > 'Z'));  // check for valid input
24
25    return drive;
26  }
```

```
1   // getFilename.cpp
2
3   /*
4     This function creates a full-path filename from the arguments passed into the function call
5
6     Overloads allow the arguments to be of the type:
7     (string, string, string, string)
8     (string, string, string)
9     (char, string, string)
10
11    The arguments are combined, in order, to form a char* containg the full-path filename
12  */
13
14  #include "f_specs.h"
15
16  using namespace std;
17
18  char* getFilename(string path, const string& sampID, const string& filter, const string& fileType) {
19
20    path += sampID + filter + fileType;                // combine each string, in order, into one string
21
22    char* filename = new char[path.length() + 1];      // allocate memory for a char* the size of the string
23    strcpy_s(filename,path.length()+1,path.c_str());   // copy the string into the char*
24
25    return filename;                                   // return the full-path filename as a char*
26  }
27
28  char* getFilename(string path, const string& filter, const string& fileType) {
29
30    path += filter + fileType;                         //combine each string, in order, into one string
31
```

```
32    char* filename = new char[path.length() + 1];         // allocate memory for a char* the size of the string
33    strcpy_s(filename,path.length()+1,path.c_str());      // copy the string into the char*
34
35    return filename;                                      // return the full-path filename as a char*
36  }
37
38  char* getFilename(const char& drive, const string& directory, const string& name){
39
40    string path = "";                                     // create an empty string
41    path += drive;                                        // add the drive letter to the string
42    path += directory;                                    // add the directory to the string
43    path += name;                                         // add the filename to the string
44
45    char* filename = new char[path.length() + 1];     // allocate memory for a char* the size of the string
46    strcpy_s(filename, path.length()+1,path.c_str());  // copy the string into the char*
47
48    return filename;                                      // return the full-path filename as a char*
49  }
```

```
1   // getID.cpp
2
3   /*
4     This function asks the user to input an identifier for the current sample.
5   */
6
7   #include "f_specs.h"
8
9   using namespace std;
10
11  string getID() {
12
13    string sampID;
14
15    cout << "Sample ID: ";                  // output request
16    getline(cin,sampID);                   // input ID
17    cin.sync();                            // clear any unwanted characters from I/O stream buffer
18    cout << endl;
19
20    for(uint16_t i=0; i<sampID.length(); ++i){
21      if ((sampID.at(i) >= 97)&&(sampID.at(i) <= 122))        // converts all alphabetic characters to
22        sampID.at(i) -= 32;                                   // upper case
23    }
24
25    return sampID;
26  }
```

```
1   // getPath.cpp
2
3   /*
4     This function combines separate pieces of a path into one string
5
6     overloading allows this function to take arguments:
7     (char, string, string)
8     (char, string)
9   */
10
11  #include <ShlObj.h>           // required to create directories in Windows
12  #include "f_specs.h"
13
14  using namespace std;
15
16  string getPath(const char& drive, const string& directory, const string& sampID) {
17
18    string path;              // create an empty string for the path
19
20    path = drive + directory;   // add the drive letter and the common directory to the path
21    SHCreateDirectoryExA(NULL,path.c_str(),NULL);  // create the directory on the filesystem
22                                        // if it does not exist
23    path += sampID;        // add the sample ID to the path
24    path += "\\";
25    while (!CreateDirectoryA(path.c_str(),NULL)) {       // create the directory for the current sample
26      int i = path.length();                        // if this directory already exists, append "_COPY"
27      path.at(i-1) = '_';                           // to the directory name, and retry.
28      path += "COPY\\";                             // this prevents data from being overwritten
29    }
30
31    return path;                                  // return the full path
```

```
32  }
33
34  string getPath(const char& drive, const string& directory) {
35
36    string path;                              // create an empty string for the path
37    path = drive + directory;                 // combine the drive letter and the directory to form the path
38    SHCreateDirectoryExA(NULL,path.c_str(),NULL);     // create the directory in the filesystem
39                                              // if it does not already exist
40    return path;                              // return the full path
41  }
```

```
1   // getPort.cpp
2
3   /*
4     This function prompts the user to enter a port name to use
5     as a serial communications port with the FW102B filter wheel
6
7     COM4 is commonly used, but check your hardware configuration to be sure
8   */
9
10  #include "f_specs.h"
11
12  using namespace std;
13
14  string getPort() {
15
16    string port;                          // allocate memory for the port name
17
18    cout << "Port name (case sensitive): ";     // prompt the user for input
19    getline(cin,port);                      // read user input
20    cin.sync();                           // discard any remaining Bytes in the I/O stream
21
22    return port;                          // return the port name
23  }
```

```
1   // image.cpp
2
3   /*
4     This is the class implementation file for the user defined class "image"
5
6     This class was designed to hold a single 12-bit image acquired by the Alpha-NIR
7     camera.  Also stored is a filename associated with the image.
8   */
9
10  #include "image.h"
11
12  image::image(const int& r, const int& c) : _rows(r),       // default constructor
13                                     _cols(c),       // default values: r=256 c=318
14                                     len(0) {        // filename is empty
15    filename = new char[len + 1];                          // data is initalized as r*c
16    filename[len] = 0;                                  // saturated 16-bit integers
17
18    data = new uint16_t[_rows * _cols];
19    for(int i=0; i<(r*c); ++i)
20      data[i] = 65535;
21  }
22
23  image::image(const image& o) : _rows(o._rows),             // copy constructor
24                             _cols(o._cols),            // copies a variable of type "image"
25                             len(o.len) {               // into new memory
26    filename = new char[len+1];
27    for(int i=0; i<len; ++i)
28      filename[i] = o.filename[i];
29    filename[len] = 0;
30
31    data = new uint16_t[_rows * _cols];
```

```
32      for(int i=0; i<o.size(); ++i)
33        data[i] = o.data[i];
34    }
35
36    image::~image() {                    // destructor
37      delete [] filename;                // releases memory that is no longer in use
38      delete [] data;
39    }
40
41    int image::rows() const{             // returns the number of rows of pixels
42      return _rows;
43    }
44
45    int image::cols() const{             // returns the number of columns of pixels
46      return _cols;
47    }
48
49    int image::size() const{             // returns the number of pixels
50      return (_rows * _cols);
51    }
52
53    char* image::get_filename() const{     // returns the filename associated with the image
54      return filename;
55    }
56
57    void image::set_filename(const char* str) {    // sets the filename associated with the image to
58      delete [] filename;                          // the variable passed into the function call
59      len = strlen(str);
60      filename = new char[len + 1];
61      for(int i=0; i<len; ++i)
62        filename[i] = str[i];
```

```
63      filename[len] = 0;
64  }
```

```
1   // image.h
2
3   /*
4     This is the class declaration file for the user defined class "image"
5
6     For details about each function, please see the class implementation file (image.cpp)
7   */
8
9   #ifndef IMAGE_H
10  #define IMAGE_H
11
12  #include <cstdint>
13  #include <string>
14
15  class image{
16  public:
17    image(const int& r = 256,
18          const int& c = 318);
19    image(const image&);
20    ~image();
21
22    int rows() const;
23    int cols() const;
24    int size() const;
25
26    char* get_filename() const;
27    void set_filename(const char*);
28
29    uint16_t* data;   // the values of the pixels in the image
30
31  private:
```

```
32      char* filename;   // the filename associated with the image
33      int _rows;        // the number of rows of pixels
34      int _cols;        // the number of columns of pixels
35      int len;          // the length of the filename
36    };
37    #endif
```

```
1   // loadAdjustMap.cpp
2
3   /*
4     this function loads the pixel offset maps created by the calibration program.
5
6     There is a different calibration file for each optical filter in use.
7   */
8
9   #include "f_specs.h"
10
11  using namespace std;
12
13  vector<dblImage> loadAdjustMap(const char& drive){
14
15    string path = getPath(drive, dirCALIBRATION);              // Get the location of the calibration files
16    vector<dblImage> offset;                                   // Empty vector to contain the offset maps
17
18    for(int i=0; i<NO_FILTERS; ++i){                           // loop once for each filter
19
20      ifstream infile;
21
22      char* filename = getFilename(path, FILTERS[i], ".ADJ");   // get the filename of the calibration file for the
23                                                               // filter in the current iteration of the loop
24      dblImage temp;                                           // holder for the offset map
25      infile.open(filename, ios::binary);                      // open the file as binary data
26      for(int i=0; i<temp.size(); ++i)
27        infile.read((char*)&temp.data[i],sizeof temp.data[i]);  // read the entire file, one double precision value at a time
28                                                               // and store in the holder
29      infile.close();                                          // close the file when done
30      offset.push_back(temp);                                  // store the data in the holding variable into the vector
31    }
```

```
32      return offset;                  // return the vector containing the offset maps
33  }
```

```
1   // openPort.cpp
2
3   /*
4     This function opens serial communications with the port specified by the argument
5     passed into the function call
6
7     If the port is successfully opened, the function returns the handle.
8     If the port cannot be opened, the function returns 0;
9   */
10
11  #include "f_specs.h"
12
13  using namespace std;
14
15  HANDLE openPort(const string& name) {
16
17    HANDLE hPort;     // port handle to be returned
18    DCB PortDCB;      // contains the DCB structure for the COM port parameters
19
20    hPort = CreateFileA(name.c_str(), GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
21                                          // Set the port as READ/WRITE
22
23    SecureZeroMemory(&PortDCB,sizeof(DCB));               // set all bits of the DCB structure to 0
24    PortDCB.DCBlength = sizeof (DCB);                     // define the length of the DCB
25
26    GetCommState(hPort,&PortDCB);                         // retrieve the current DCB settings from hPort
27
28    PortDCB.BaudRate = 115200;                          // set the Baud Rate to 115200
29    PortDCB.ByteSize = 8;                             //        Byte Size to 8
30    PortDCB.Parity = NOPARITY;                          //        Pairity = NONE
31    PortDCB.StopBits = ONESTOPBIT;                        //        Stop Bit = 1
```

```
32      PortDCB.fRtsControl = RTS_CONTROL_DISABLE;              //      Flow Control = none
33      PortDCB.fDtrControl = DTR_CONTROL_DISABLE;
34      PortDCB.fAbortOnError = FALSE;
35      PortDCB.fBinary = TRUE;
36      PortDCB.fParity = FALSE;
37
38      PortDCB.fOutxCtsFlow = FALSE;
39      PortDCB.fOutxDsrFlow = FALSE;
40      PortDCB.fDsrSensitivity = FALSE;
41      PortDCB.fTXContinueOnXoff = TRUE;
42      PortDCB.fOutX = FALSE;
43      PortDCB.fInX = FALSE;
44      PortDCB.fErrorChar = FALSE;
45      PortDCB.fNull = FALSE;
46
47      if (!SetCommState (hPort, &PortDCB))                    // Send the current DCB structure to hPort
48        return FALSE;                                 // return 0 if COM port does not respond
49      return hPort;                                 // return port handle if successfuly opened
50    }
```

```
1  // processImages.cpp
2
3  /*
4    This function processes the images passed to it in the function call
5
6    The process includes:
7    pixel clasification
8    creating a binary image (mask) based on pixel classification
9    Applying the mask to different images of the same object
10   Averaging values in each image of pixels that correspond to a 'true' value in the the mask
11   Storing a summary of the data (Sample ID, Used Pixel count, Average reflectance at each waveband,
12     and classification method used
13   Storing a copy of the mask for visual inspection
14 */
15
16 #include "f_specs.h"
17 #include <iomanip>
18
19 using namespace std;
20
21 vector<double> processImages(const vector<dblImage>& I, const string& ID){
22
23   vector<double> values;                              // Vector to hold the average pixel values
24
25   binImage maskImg = createMask(I.at(0),I.at(1),1.308,1.355);   // creates the binary classification mask
26
27   for(uint8_t i=2; i<(I.size()-1); ++i){
28     double sum = 0;                          // cumulative sum of used pixel values
29     double temp;                            // holder for the calculated average
30     for(int j=0; j<I.at(i).size(); ++j){          // loops through every pixel
31       if (maskImg.data[j])
```

```
32              sum += I.at(i).data[j];                  // adds the current pixel value to the sum
33            }                                        // if the pixel corresponds to a 'true' in the mask
34            temp = sum / (double)maskImg.count();           // cumulative sum divided by the number of 'true' pixels
35            values.push_back(temp);                      // stores the calculated average for late use
36          }
37
38          string filename = maskImg.get_filename();          // gets the filename for the sample
39          int len = filename.length();
40          filename.erase(len-3);
41          filename += "TXT";                              // changes the file extension to .TXT
42
43          ofstream outfile;
44
45          /*********** The Summary File for the sample is created ******************/
46
47          outfile.open(filename.c_str());                      // opens a file using the filename provided
48          outfile << fixed << setprecision(3);                 // outputs all numbers to 3 decimal places
49          outfile << "ID: " << ID << endl;                        // writes the ID to the file
50          outfile << "Pixels Used: " << maskImg.count() << endl;      // writes the pixel count
51          outfile << "1450: " << values.at(0) << endl;                  // writes the 1450 nm average
52          outfile << "1550: " << values.at(1) << endl;                  // writes the 1550 nm average
53          outfile << "1600: " << values.at(2) << endl;                  // writes the 1600 nm average
54          outfile << "Method: 650/1300 - 1.308 : 1.355 thresholding" << endl;     // writes the classification method
55          outfile.close();                                      // closes the file
56
57          /************  The binary classification mask is saved ******************/
58
59          outfile.open(maskImg.get_filename());        // opens a file using the filename provided
60          int x = maskImg.rows();                   // x = number of rows of pixels
61          for(int i=0; i<maskImg.size(); ++i){         // loop through every pixel
62            outfile << maskImg.data[i];               // output to file: 1 for 'true' or 0 for 'false'
```

```
63        if ((i % x) == (x - 1))              // if the current pixel is the last pixel in the row
64          outfile << endl;                   // output a newline character
65        else
66          outfile << " ";                    // otherwise output a single space
67      }
68      outfile.close();                       // close the file when done
69
70      return values;                         // returns the average pixel values
71    }
```

```
1  // saveBufferBin.cpp
2
3  /*
4    This function saves an image buffer from the Alpha-NIR camera as binary data.
5
6    The data is written as continuous Bytes representing the pixel values.
7    Each value written will be sizeof(double) Bytes long.  No other delimeter is used in the file.
8
9    When reading the binary file produced by this function, make sure that all of the bytes are read
10   for each value.
11 */
12
13 #include "f_specs.h"
14
15 using namespace std;
16
17 void saveBufferBin(const dblImage& I){
18
19   ofstream outfile;
20
21   string filename = I.get_filename();          // Get the filename for the sample passed
22                                                // into the function
23   int len = filename.length();
24   filename.erase(len-3);
25   filename += "BIN";                           // replace the file extension with .BIN
26
27   outfile.open(filename.c_str(), ios::binary);      // open a file with the provided filename as binary data
28   for(int i=0; i<I.size(); ++i)                     // loop through every pixel in the image
29     outfile.write((char*)&I.data[i], sizeof I.data[i]);    // write the Bytes representing the pixel value
30   outfile.close();                                  // close the file when done
31 }
```

```
1   // saveImgDec.cpp
2
3   /*
4     This function saves the image passed into it as a readable data file.
5
6     The value for each pixel is separated by a space in the row, while each
7     row is separated by a newline character.
8
9     The file created by this function can be read directly into a numerical analysis program
10    that will read delimited files.  The column delimiter is a single space.
11  */
12
13  #include "f_specs.h"
14  #include <iomanip>
15
16  using namespace std;
17
18  void saveImgDec(const dblImage& I){
19
20    ofstream outfile;
21    string str = I.get_filename();            // get the filename from the image
22    int len = str.length();
23    str.erase(len-3);
24    str += "DAT";                          // replace the file extension with .DAT
25
26    outfile.open(str.c_str());               // open the file
27
28    outfile << fixed << setprecision(3);      // output all values to 3 decimal places
29    for (int i=0; i<I.size(); ++i){           // loop through every pixel
30      outfile << I.data[i];
31      if((i % I.cols()) == (I.cols() - 1))    // if the pixel was the last in the row
```

```
32          outfile << endl;                  // write a newline character
33        else
34          outfile << " ";                   // otherwise write a single space
35      }
36      outfile.close();                      // close the file when done
37    }
```

```
1   // setIntegrationTime.cpp
2
3   /*
4     This function sets the integration time of the image sensor in the Alpha-NIR
5     camera.
6
7     sid is a valid, open session ID
8
9     t is an integer corresponding to an integration time.
10
11    The valid values for t are between 53 and 51350, inclusive.
12    The range of t is dependent on the selected integration mode.
13    This function is written for the 'Normal Integration' Mode (INT_MODE = 0)
14
15    Integration time in microseconds = (51377.5 - t) * .65185
16  */
17
18  #include "f_specs.h"
19  #include <bitset>
20
21  using namespace std;
22
23  void setIntegrationTime(const SESSION_ID& sid, uint16_t t) {
24
25    int timeout = 500;          // value for serial timeout
26
27    if (t<53)                   // if t falls outside of the valid range [53 - 51350]
28      t = 53;                   // assign t the closest valid value
29    else if (t>51350)
30      t = 51350;
31
```

```
32
33      uint16_t sum = 0;          // allocate memory for the checksum
34      uInt32 size = 11;          // size of serial buffer to be sent
35      unsigned char* buffer = new unsigned char[size];   // allocate memory for a serial packet
36
37
38      buffer[0] = 0x49;                   // Process (Byte 1): Must be 0x49 (ASCII character 'I')
39      buffer[1] = 0x03;                   // Function (Byte 2&3): 0x0303 INT_TIMER
40      buffer[2] = 0x03;
41      buffer[3] = 0xFF;                   // Status (Byte 4): Slave response
42      buffer[4] = 0x00;                   // Packet Count (Byte 5): used for debugging
43      buffer[5] = 0x00;                   // Byte Count (Byte 6&7): number of data Bytes to be sent
44      buffer[6] = 0x02;
45      buffer[7] = t / 0x100;              // Data (Byte 8&9): desired integration time to set
46      buffer[8] = t % 0x100;
47      for(uint8_t i=0; i<(size - 2); ++i)
48        sum += buffer[i];                 // sums the preceding Bytes
49      buffer[size - 2] = sum / 0x100;       // Checksum (Byte 10&11): validates packet
50      buffer[size - 1] = sum % 0x100;
51
52      imgSessionSerialFlush(sid);                                 // clear serial I/O buffer
53      imgSessionSerialWrite(sid, (char*)buffer, &size, timeout);      // send packet to slave
54      imgSessionSerialReadBytes(sid,(char*)buffer, &size, timeout);   // read response from slave
55
56      delete [] buffer;                   // release memory that is no longer needed
57
58      size = 11;
59      sum = 0;
60      buffer = new unsigned char[size];      // allocate memory for a serial packet
61
62      buffer[0] = 0x49;                   // Process (Byte 1): Must be 0x49 (ASCII character 'I')
```

```
63      buffer[1] = 0x81;                   // Function (Byte 2&3): 0x8101 READ FPA mode
64      buffer[2] = 0x01;
65      buffer[3] = 0xFF;                   // Status (Byte 4): Slave response
66      buffer[4] = 0x00;                   // Packet Count (Byte 5): used for debugging
67      buffer[5] = 0x00;                   // Byte Count (Byte 6&7): number of data Bytes to be sent
68      buffer[6] = 0x02;
69      buffer[7] = 0x00;                   // Data (Byte 8&9): this will be overwriten by the Slave
70      buffer[8] = 0x00;
71      for(uint8_t i=0; i<size-2; ++i)
72        sum += buffer[i];                 // Sum the bytes in the packet
73      buffer[size - 2] = sum / 0x100;         // Checksum (Byte 10&11): validates the packet
74      buffer[size - 1] = sum % 0x100;
75
76      imgSessionSerialFlush(sid);                             // clear the serial I/O Buffer
77      imgSessionSerialWrite(sid, (char*)buffer, &size, timeout);      // send the packet to the slave
78      imgSessionSerialReadBytes(sid, (char*)buffer, &size, timeout);  // read the slave response
79
80      bitset<8> data(buffer[8]);          // store Byte 9 in the Slave response for bitwise use
81
82      /*
83        Data (Byte 8&9): 0b000000000000abcd
84        First 12 bits are reserved for future use: Must be low
85        Bit a: Integration capicitor.  0 = 10fF, 1 = 210 fF
86        Bit b: Integration mode.       0 = Normal, 1 = Short
87        Bit c: Orientation x.          0 = Normal, 1 = flip
88        Bit d: Orientation y.          0 = Normal, 1 = flip
89      */
90
91      if (data.at(3))
92        buffer[8] -= 8;         // Set 10fF capacitor
93      if (data.at(2))
```

```
94        buffer[8] -= 4;          // Set Normal Integration mode
95
96      sum = 0;                   // reset checksum
97      size = 11;                 // define size of packet
98      buffer[1] = 0x01;          // Function MSB (Byte 2): WRITE FPA mode
99      for(uint8_t i=0; i<size-2; ++i)
100       sum += buffer[i];              // Sum Bytes in packet
101     buffer[size-2] = sum / 0x100;    // Append new Checksum to packet
102     buffer[size-1] = sum % 0x100;
103
104     imgSessionSerialFlush(sid);                              // Clear serial I/O buffer
105     imgSessionSerialWrite(sid,(char*)buffer,&size,timeout);     // send packet to slave
106     imgSessionSerialReadBytes(sid,(char*)buffer,&size,timeout); // read response from slave
107     delete [] buffer;                                        // release memory no longer needed
108   }
```

```
1   // snapSet.cpp
2
3   /*
4     This function captures a pre-defined number of images of an object.
5
6     The number of captures is defined by a pre-processor directive in the "f_specs.h" file
7
8     The exact time between captures is left to processor speed, and communication time.
9   */
10
11  #include "f_specs.h"
12
13  using namespace std;
14
15  vector<image> snapSet(const INTERFACE_ID& iid, const SESSION_ID& sid, char* filename) {
16
17    vector<image> imageSet;              // create a vector to hold the image captures
18
19    for(int i=0; i<REPS; ++i) {
20      image I;                           // allocate memory for a single image
21      imgSnap(sid, (void **)&I.data);    // store the current sensor data
22      I.set_filename(filename);          // save filename with current image
23      imageSet.push_back(I);             // store single image in the vector
24    }
25
26    return imageSet;                     // return the vector of images
27  }
```

APPENDIX B-2

MATLAB CODE

```
function J = Apply_Mask(image,mask)
%Apply_Mask Applies a binary mask image
%   This function takes in an intensity image and a binary mask image.
%   It applies the binary mask to the intensity image and returns the masked image.

n = length(image);                % n = 81408 (256 x 318)
m = length(mask.img);             % m = 81408
if m~=n                           % error checking
    error('dimension mismatch');
end
count = 0;                        % position counter
J = zeros(mask.count,1);          % new intensity image filled with zeros
for i = 1:n                       % loop to check every pixel
    if mask.img(i)
        count = count + 1;        % increment position
        J(count) = image(i);      % translate value to masked image if corresponding binary pixel is true
    end
end
end

```

```
function J = buf2img( I )
%buf2img formats image buffer into matrix
%   This function takes a raw image array (81408 values) and formats it
%   into an image matrix (256 x 318)

J = zeros(256,318);              % preallocates image matrix
  for i=1:256
    for j=1:318
      J(i,j) = I(318*(i-1)+j);    % Translates array into matrix
    end
  end

end

```

```
function x = Calculate_Bin_Count( data )
%Calculate_Bin_Count Calculates the optimum number of histogram bins for a given distribution
%   This function takes in a distribution of data and calculates the optimum number of bins
%   for a histogram.  The optimum bin size is calculated using the function size = 2*IQR*n^(-1/3),
%   then finds the number of bins the distribution requires to meet the bin size.

k = iqr(data);                          % calculate IQR
n = length(data);

bin_size = 2*k*(n^(-1/3));                  % calculate bin size
max_val = max(data);
min_val = min(data);
x = round((max_val - min_val) / bin_size);      % calculate number of bins needed
                                    % to reach optimum bin size.
end

```

```
function img_mean = Calculate_Mean(image,mask)
%Calculate_Mean Calculates the average value of valid pixels.
%   This function takes an intensity image and a binary mask image.
%   First the mask is applied to the intensity image.
%   Then the average value of unmasked pixels is calculated and returned

I = Apply_Mask(image,mask);    % binary mask is applied
img_mean = mean(I);            % average of unmasked pixels is calculated
end
```

```
1  function S = Empty_Sample()
2  %Empty_Sample Preallocates memory for a cotton sample
3  %   This function preallocates teh memory that will be needed for a single cotton sample.
4
5  S = struct('SampID',[],'mask',[],'mean_1450',[],'mean_1550',[],'mean_1600',[],'mic',[]);
6  end
```

```
function data = Empty_Set( count )
%Empty_Set Preallocates memory for a set of cotton samples
%   This function preallocates memory needed for an array with
%   the size 'count' of cotton samples

mask = struct('img',[],'count',[]);        % creates a structure to store both the mask image
                                    % and the # of pixels with a true value
mask.img = false(81408,1);                % fills the mask image with false values
mask.count = 0;                         % initalizes the mask count to zero

data = Empty_Sample();                    % creates an empty data set

data.SampID = cell(count,1);              % allocates memory for the sample ID's

data.mean_1450 = zeros(count,1);          % allocates memory for the average
data.mean_1550 = zeros(count,1);          % pixel values from each filter
data.mean_1600 = zeros(count,1);          % by initalizing each to zero
data.mic = zeros(count,1);                % allocates memory for the micronaire values

data.mask = repmat(mask,count,1);         % allocates memory for the image masks using the structure created earlier


end

```

```
function S = Find_Optimum( method, ctn_stage, plow, phigh, rlow, rhigh )
%Find_Optimum Finds the combination of parameters that yields the best R^2 value
%   Takes in parameter ranges to pass to the Run_Method function.
%   compares R^2 values and returns the parameters that provide the best relationship


 images = cellstr(['650 ';...            % variable containing strings for the filter bands
                   '1300';...
                   '1450';...           % These strings are used to specify which filtered image
                   '1550';...           % is being used in the Single_Mode_Method function (Line 53,54)
                   '1600']);            % for pixel classification

 if (strcmp(method,'Ratio'))            % selects Image Ratio method
   S = struct('rsqr', 0,...             % structure containing "best" statistics  and parameter data
      'Fstat', {},...
      'Pval', {},...
      'err', {},...
      'percentage', {},...
      'ratio', {});
   S(1).rsqr = 0;                       % initalize "best" R-squared to zero
   for x= plow:.01:phigh                % loop for percentage values
     for y= rlow:.1:rhigh               % loop for ratio values
       [data count] = Run_Method('Ratio_Mode',...
                          ctn_stage, x, y); % runs the method using the current iteration parameters
       Print_List(['S:\RESULTS\' method '_' ctn_stage '_' num2str(x*100)...
                '_' num2str(y*10) '.TXT'],data);      % outputs the method data to a file
       X = [ones(count,1) data.mean_1450 data.mean_1550 data.mean_1600]; % independent variables used for linear model
       Y = data.mic;                    % dependent variable used for linear model
       [b,bint,r,rint,stats] = regress(Y,X); % linear regression to find an equation
                                  % for micronaire in terms of mean reflectance
       if (stats(1) > S.rsqr)           % Check to see if the R-squared value improved
```

```
            S.rsqr = stats(1);                % stores current results and parameters as "best"
            S.Fstat = stats(2);
            S.Pval = stats(3);
            S.err = stats(4);
            S.percentage = x;
            S.ratio = y;
          end
        end
      end
    else if (strcmp(method,'Single'))       % selects Single Image method
      S = struct('rsqr', {},...             % structure containing "best" statistics and parameter data
                 'Fstat', {},...
                 'Pval', {},...
                 'err', {},...
                 'percentage', {},...
                 'ratio', {},...
                 'image', {});
      S(1).rsqr = 0;                        % initalize "best" R-squared to zero
      for i = 1:5                           % loop for image names
        for x= plow:.01:phigh               % loop for percentage values
          for y= rlow:.1:rhigh              % loop for ratio values
            [data count] = Run_Method('Ratio_Mode',...
                        ctn_stage, x, y, images(i)); % runs the method using current iteration parameters
            Print_List(['S:\RESULTS\' method '_' ctn_stage '_' num2str(x*100) '_'...
                      num2str(y*10) '_' char(images(i)) '.TXT'],data);     % outputs the method data to a file
            X = [ones(count,1) data.mean_1450...
                data.mean_1550 data.mean_1600];  % independent variables used for linear model
            Y = data.mic;                       % dependent variable used for linear model
            [b,bint,r,rint,stats] = regress(Y,X); % linear regression to find an equation
                                                % for micronaire in terms of mean reflectance
            if (stats(1) > S.rsqr)              % check to see if the R-squared value improved
```

```
63              S.rsqr = stats(1);              % stores current results and parameters as "best"
64              S.Fstat = stats(2);
65              S.Pval = stats(3);
66              S.err = stats(4);
67              S.percentage = x;
68              S.ratio = y;
69              S.image = images(i);
70            end
71          end
72        end
73      end
74    end
75    end
76  end
```

```
1   function mask = Generate_Mask( lower, upper, I )
2   %Generate_Mask creates a binary mask for an image based upon a given threshold
3   %   This function takes in an image and an upper and lower boundary for pixels
4   %   to classify as "cotton".  Pixels that do not fall between the boundaries are
5   %   given a value of 'false' while pixels that fall within the boundaries, inclusive,
6   %   are given a value of 'true'.
7
8   mask.img = false(size(I));                  % preallocate memory for the binary image
9   n = length(I);                          % n = 81408
10  mask.count = 0;                           % initalize the number of pixels used to zero
11  for i=1:n
12     if (I(i) >= lower) && (I(i) <= upper)    % check to see if the pixel value is within the boundaries
13        mask.img(i) = true;                 % set corresponding mask pixel to 'true'
14        mask.count = mask.count + 1;        % increment counter
15     end
16  end
17  end
18
```

```
function mode_index = hist_mode(H)
%hist_mode finds the index value of the histogram mode
%   This function takes in a histogram and finds the index value of
%   the most frequenty occuring value in a distribution.  If highest rate of occurance
%   appears in two or more bins, the index with the lowest numerical value is returned.

n = length(H);              % n = number of histogram bins
value = 0;                  % initalize value as zero
mode_index = 0;             % initalize the mode index as zero
for x = 2:(n-1)             % loop through each frequency bin excluding the first and last
    if H(x) >= value        % determine if the current bin contains more values than the previously stored
        value = H(x);       % store the number of values for later comparison
        mode_index = x;     % store current index as "mode"
    end
end

end

```

```
function [A B C D E] = Load_Cotton_Sample( ctn_stage, Samp_ID )
%Load_Cotton_Sample loads data for a given sample ID at a given cotton stage
%   This function loads image data for a given cotton sample (Samp_ID)
%   at a given cotton stage (seed or lint) from the stored binary files

filename = ['S:\COMMON\' ctn_stage '\' Samp_ID '\' Samp_ID '_650.BIN'];       % filename for 650nm image
A = Read_Cotton_Image(filename);                                              % read 650nm image
filename = ['S:\COMMON\' ctn_stage '\' Samp_ID '\' Samp_ID '_1300.BIN'];      % filename for 1300nm image
B = Read_Cotton_Image(filename);                                              % read 1300nm image
filename = ['S:\COMMON\' ctn_stage '\' Samp_ID '\' Samp_ID '_1450.BIN'];      % filename for 1450nm image
C = Read_Cotton_Image(filename);                                              % read 1450nm image
filename = ['S:\COMMON\' ctn_stage '\' Samp_ID '\' Samp_ID '_1550.BIN'];      % filename for 1550nm image
D = Read_Cotton_Image(filename);                                              % read 1550nm image
filename = ['S:\COMMON\' ctn_stage '\' Samp_ID '\' Samp_ID '_1600.BIN'];      % filename for 1600nm image
E = Read_Cotton_Image(filename);                                              % read 1600nm image

end

```

```
function data = Load_Mic( filename )
%Load_Mic loads the micronaire values from a file
%   This function loads the measured micronaire values for each sample
%   from a text file

fid = fopen(filename);                  % open the file
s = fgetl(fid);                         % get first line from file (number of lines of data)
fgetl(fid);                             % throw away second line (headings)
n = str2double(s);                      % change first line into a number
data = struct('ID',[],'mic',[]);        % structure to hold data read
data.ID = cell(n,1);                    % preallocate memory for sample IDs
data.mic = repmat(0,n,1);               % preallocate memory for micronaire data
count = 0;                              % counter for number of values read
while ~feof(fid);                       % read until the file ends
    c = fscanf(fid,'%c',1);             % read a single character from the file
    str = '';                           % initalize an empty string
    while ~(c==' ')                     % loop while true
        str = [str c];                  % add the character read to str
        c = fscanf(fid,'%c',1);         % read the next character frmo the file
    end
    count = count + 1;                  % increment counter by one
    data.ID(count) = cellstr(str);      % store str as the current sample ID
    data.mic(count) = fscanf(fid,'%f',1);   % read in a floating point value for the current micronaire
    fgetl(fid);                         % throw away the rest of the line
end
fclose(fid);                            % close file when done
end
```

```
function count = Print_List( filename,data )
%Print_List creates a file containing data calculated by the Run_Method function
%   This function creates a text file containing data given to it.
%   The data is in the form of a structure of arrays created by the Run_Method function
%   The data is stored in a file with the name given in the filename variable

%   count contains the number of bytes written to the file.


n = length(data.SampID);                        % n = number of samples
fid = fopen(filename,'wt');                     % open file with write privledges
count = fprintf(fid,'ID f1450 f1550 f1600 mic\n');      % print the column headings to file
for i=1:n                                       % loop for each sample
   m1450 = data.mean_1450(i);
   m1550 = data.mean_1550(i);
   m1600 = data.mean_1600(i);
   mic = data.mic(i);
   SampID = char(data.SampID(i));
   count = count + fprintf(fid,[SampID ' %8.3f %8.3f %8.3f %3.2f\n'],...
                   [m1450,m1550,m1600,mic]);   % print data for current sample to file
end
fclose(fid);                                    % close file when done
end

```

```
function [mask m1450 m1550 m1600] = Ratio_Mode_Method( ctn_stage, SampID, perc, ratio )
%Raio_Mode_Method performs the ratio mode method on a single cotton sample using the specified parameters
%   This function generates a binary mask image, and calculates the average values for
%   each NIR filter for a single cotton sample.

  [A B C D E] = Load_Cotton_Sample(ctn_stage,SampID);  % Loads the five images taken during data collection
  n = length(A);                                % n = 81408
  img_ratio = zeros(n,1);                       % preallocates memory for the ratio image
  for i=1:n
    img_ratio(i) = A(i) / B(i);                 % calculates the ratio image from the 650 and 1300 nm images
  end
  bin_count = Calculate_Bin_Count(img_ratio);         % calculates the optimum number of bins for the distribution
                                              % of the ratio image
  [img_hist img_bin] = hist(img_ratio,bin_count);      % generates a histogram of the distribution of the ratio
                                              % image with a specified number of bins
  [lower_bin upper_bin] = Set_Bound_1(img_hist, perc, ratio);     % generates the boundaries of the classification
                                                % threshold in terms of bin number
  upper_val = img_bin(upper_bin);                   % translates bin number into pixel value
  lower_val = img_bin(lower_bin);
  mask = Generate_Mask(lower_val, upper_val, img_ratio); % generates a binary mask image using the classification
                                                %threshold boundaires
  m1450 = Calculate_Mean(C,mask);                   % calculates average values for each of the NIR images after
  m1550 = Calculate_Mean(D,mask);                   % being masked
  m1600 = Calculate_Mean(E,mask);
end

```

```
function Val = Read_Cotton_Image( filename )
%Read_Cotton_Image reads binary image data
%   This function reads binary image data from a file and formats it into
%   useable matlab data

fid = fopen(filename);          % open the specified file
Val = fread(fid, 'double');     % read data from file as double precision
fclose(fid);                    % close file when done
end

```

```
function [data count] = Run_Method( method, ctn_stage, var1, var2, var3 )
%Run_Method runs the specified method with the specified parameters
%   This method runs the method specified by the user, at the stage specified by the user
%   with the parameters specified by the user.

mic_data = Load_Mic('S:\COMMON\MIC.TXT');  % load the micronaire data from file
fid = fopen(['S:\COMMON\' ctn_stage '\'...
        ctn_stage '_84.TXT']);        % open the list of samples for the specified stage
str = fgetl(fid);                     % read number of samples from file
count = str2double(str);              % convert to number
data = Empty_Set(count);              % preallocate memory for the number of samples expected
n = uint8(0);                         % initalize 8-bit unsigned integer as zero
while ~feof(fid)                      % loop while not at the end of the file
   SampID = fgetl(fid);               % read the sample ID from file
   n = n + 1;                         % increment counter
   switch method                      % use specified method
      case 'Ratio_Mode'
         [data.mask(n) data.mean_1450(n) data.mean_1550(n) data.mean_1600(n)]...
         = Ratio_Mode_Method(ctn_stage,SampID,var1,var2);
      case 'Single_Mode'
         [data.mask(n) data.mean_1450(n) data.mean_1550(n) data.mean_1600(n)]...
         = Single_Mode_Method(ctn_stage,SampID,var1,var2,var3);
      otherwise
         error('Unknown Method')    % invalid method chosen
   end
   data.SampID(n) = cellstr(SampID);  % store SampID with sample data
end
fclose(fid);                          % close file when done
m = length(mic_data.ID);
for i=1:n                             % loop to iterate through stored data
   str = char(data.SampID(i));        % str = current sample's ID
```

```
32      A = str(1:3);                   % A = first three characters of ID
33      for j=1:m                       % loop to search through micronare data
34          str = char(mic_data.ID(j));    % str = current micronaire value ID
35          B = str(1:3);               % B = first three characters of ID
36          if strcmpi(A,B)               % compare A and B
37              data.mic(i) = mic_data.mic(j); % if A=B store the micronaire data for the current sample
38              break                   % break out of search loop to next iteration of data loop
39          end
40      end
41  end
42  end
```

```
function [low high] = Set_Bound_1( H, perc, ratio )
%Set_Bound_1 sets the classification threshold boundaries for a histogram
%   This function sets the pixel classification boundaries based on the
%   given parameters

n = sum(H);                 % sum the frequency values (n = 81408)
count = 0;                  % initalize counter
low = uint16(0);            % start low boundary at 0
while (count < (n*perc))    % loop to include specified percentage of pixels
    low = low + 1;          % increment index number of low boundary
    count = count + H(low);   % add bin frequency to count
end
m = hist_mode(H);           % find bin number of histogram mode
high = uint16(0.5 + m + ratio*(m-low));   % determine high boundary using ratio,
                                %mode, and low boundary
end

```

```
function [low high] = Set_Bound_1b( H, perc, ratio )
%Set_Bound_1b sets classification threshold boundaries
%   This function sets the classification boundaries of a histogram
%   based on a specified percentage of pixel exclusion and a ratio of
%   boundary distance to the mode

n = 81408;                  % total pixel count
count = 0;                  % initalize counter
high = uint16(length(H)+1);     % initalize high boundary
while (count < (n*perc))        % loop while the boundary doesn't exclude enough pixels
    high = high - 1;            % decrement the boundary value by one
    count = count + H(high);    % add the bin frequency to the excluded pixel count
end
m = hist_mode(H);               % find the bin number for the distribution mode
low = uint16(0.5 + m - ratio*(high-m));     % set the lower boundary using the mode, ratio,
                                % and high boundary
end

```

```
function [mask m1450 m1550 m1600] = Single_Mode_Method( ctn_stage, SampID, perc, ratio, img )
%Single_Mode_Method runs the single mode method for a single cotton sample
%   This function classifies pixels in a single image and calculates the average value
%   of NIR reflectance in three wavebands of a given cotton sample

  [A B C D E] = Load_Cotton_Sample(ctn_stage,SampID);             % load image data from binary files
  switch img                                                      % select the single image specified
     case '650'                                                   % to create the binary mask
        I = A;
     case '1300'
        I = B;
     case '1450'
        I = C;
     case '1550'
        I = D;
     case '1600'
        I = E;
     otherwise
        error('invalid image');          % invalid image specified
  end
  bin_count = Calculate_Bin_Count(I);      % calculates the optimum number of bins
  [img_hist img_bin] = hist(I,bin_count);  % generates a histogram for the specified image using a specified bin count
  [lower_bin upper_bin] = Set_Bound_1b(img_hist, perc, ratio);
                                % generates classification threshold boundaries based on given parameters
  if lower_bin == 0                     % error correction.  Bin 0 does not exist.
     lower_bin = 1;
  end
  upper_val = img_bin(upper_bin);          % converts bin number to pixel value
  lower_val = img_bin(lower_bin);          % converts bin number to pixel value
  mask = Generate_Mask(lower_val,...       % generates binary mask image based on classification threshold boundaries
                       upper_val, I);
```

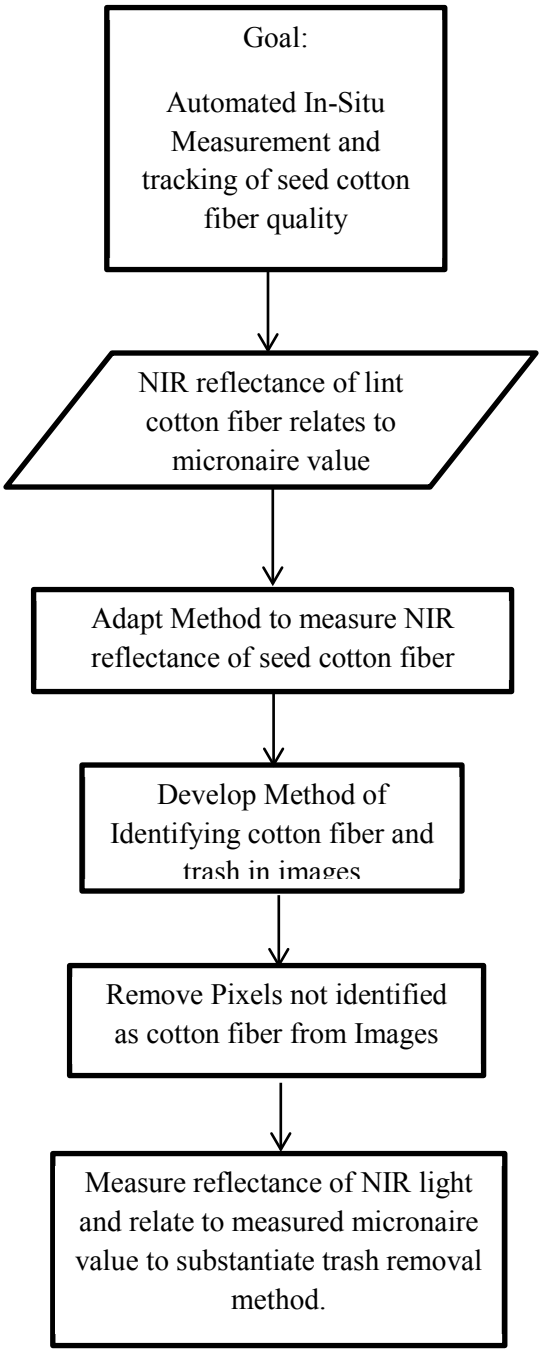
```
32      m1450 = Calculate_Mean(C,mask);         % applies the binary mask to the NIR images and
33      m1550 = Calculate_Mean(D,mask);         % calculates the average pixel value for each
34      m1600 = Calculate_Mean(E,mask);
35
36    end
37
```

APPENDIX C

FLOW CHARTS

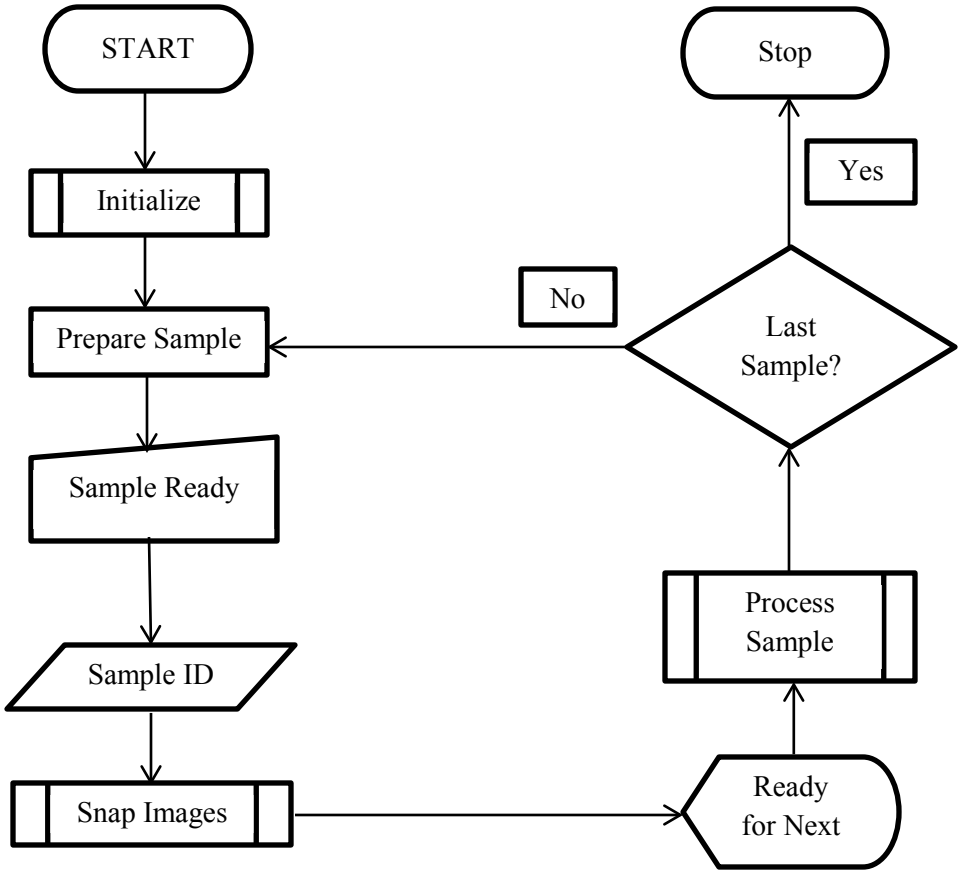
APPENDIX C-1: PROJECT DESIGN FLOW CHART

This flow chart illustrates the prototype system design process. Starting with a goal of measuring the quality of seed cotton and adapting existing research on lint cotton, methods of measuring seed-cotton fiber quality were developed and tested.



APPENDIX C-2: PROGRAM FLOW CHART

This flow chart illustrates the flow of the prototype program. When the program is run, it first goes through an initialization process, reading calibration data from a file and allowing the QTH lamps to reach a steady temperature. Once initialization is done, the first sample is prepared for data collection. This involves collecting cotton and pressing it against the sample window. The system waits until a signal is given that the sample is ready for data collection, and a unique identifier is generated for the sample. In the prototype, the signal for the sample being ready was the operator input of the sample ID in the console. Next a series of images are acquired through different optical filters. When image acquisition is complete, the system announces that it is ready for the next sample to be prepared. The program then processes the images it had just taken while waiting for the next sample to be ready, and raises the logical question, "Was that the last sample?" If that was not the last sample, the program loops back and waits until the sample collection system indicates that another sample is ready to be processed. When the last sample comes through, the system finishes any remaining processes, closes any open files, and halts.



VITA

Name: Vincent Paul Schielack III

Address: 214 Agricultural Hall,
Oklahoma State University
Stillwater, OK 74078

Email Address: vince.schielack@aggienetwork.com

Education: B.S., Biological and Agricultural Engineering, Texas A&M University, 2006
M.S., Biological and Agricultural Engineering, Texas A&M University, 2011